Integrated Systems and Software Engineering Process

SPC-96001-CMC
Version 01.00.04

May 1996

Integrated Systems and Software Engineering Process

SPC-96001-CMC

Version 01.00.04

May 1996

Susan Rose
Lisa Finneran
Sanford Friedenthal
Howard Lykins
Peter Scott

Produced by the
SOFTWARE PRODUCTIVITY CONSORTIUM

SPC Building
2214 Rock Hill Road
Herndon, Virginia 22070

CONTENTS

FIGURES

TABLES

This page intentionally left blank.

ACKNOWLEDGMENTS

The Software Productivity Consortium wishes to recognize the following contributors to the development of this technical report:

- The Integrated Systems and Software Engineering Process (ISSEP) development team: Lisa Finneran, Sanford Friedenthal, Howard Lykins, Susan Rose, and Peter Scott.
- Our external reviewers: Perry R. DeWeese (Lockheed Martin Aeronautical Systems), Ken Jackson (Requirements Engineering Limited), and Thomas J. Pighetti (Lockheed Martin Astronautics) for their comments and suggestions.
- Our internal reviewers: Christine Ausnit, Tim Powell, and Sarah Sheard for their guidance.
- Tim Powell for his role as program manager.
- Lisa Finneran for her support as project manager.
- The Generic Systems Engineering Process (GSEP) development team for creation of the GSEP concepts and activities which have been incorporated into the ISSEP model.
- The Engineering Software-Intensive Systems (ESIS) Users Group for helping the development team to focus on areas of concern in the systems and software development process.
- Dennis M. Buede, George Mason University and Wolter J. Fabrycky, Scott F. Midkiff, and André T. Ramos, Virginia Polytechnic Institute and State University, for their insights on application of the GSEP model which provided a greater understanding of the ISSEP model's underlying concepts.
- Bobbie Troy for her technical editing and Debbie Morgan for her word processing assistance.

This page intentionally left blank.

EXECUTIVE SUMMARY

This technical report describes the Integrated Systems and Software Engineering Process (ISSEP) created by the Software Productivity Consortium (the Consortium). ISSEP's purpose is to enable improvement of the overall systems development process allowing systems and software engineers to more efficiently perform their work. The ISSEP model accomplishes this goal by defining a set of management and technical activities, and most importantly, defining the mechanisms to coordinate and control the development effort. The ISSEP model integrates the set of management and technical development activities, incorporates risk management activities, and complies with major systems and software engineering standards.

Balanced Approach

The ISSEP model provides a balanced process that equally emphasizes the management and technical perspectives. Management activities provide the control necessary for developing a system. The technical activities define both the systems and software development activities. A balanced approach ensures that management is provided the technical expertise necessary for decision making and that the engineers are provided the plans and procedures for meeting customer, user, and organizational expectations.

Information Flow

The ISSEP model focuses on information flow and identification of critical systems and software engineering process interfaces. The information flow defines the coordination and communication mechanisms necessary for a successful system/software delivery. The ISSEP model defines the minimum set of required interfaces between management and technical activities, among management activities, and among the set of technical activities. These interfaces either provide the necessary information to perform an activity or provide feedback information necessary to identify and mitigate risk.

Standards Compliance

There are several emerging standards, both military and commercial, that have direct impact on today's system and software development efforts. Systems/software acquisitions require that industry be compliant and tailor many of these standards. Therefore, any process industry adopts must comply with these standards. The ISSEP model provides a high level process framework for implementing these standards, by defining activities and information flows for integrating the requirements documented in these standards.

Adaptable Process

The ISSEP model is an adaptable process and, as such, can be applied to a wide variety of systems and software development efforts. The ISSEP model is tailorable and can be scaled to a wide variety of development efforts.

1. INTRODUCTION

1.1 OVERVIEW

For the past several years, the Software Productivity Consortium (the Consortium) member companies, as well as the software industry as a whole, have identified the need for integrated systems and software engineering development processes and methods. The Consortium has gathered information concerning this topic from several meetings, workshops (Software Productivity Consortium 1993b, 1994a, 1995a), and national and international groups (Office of Naval Research 1994; NCOSE 1995). These needs cover a broad range, including the lack of integrated/seamless development methods, the lack of management visibility into the technical activities, and excessive numbers of hardware and software incompatibilities discovered at integration time. Using this information as input, this technical report describes an Integrated Systems and Software Engineering Process (ISSEP), including the management and technical development activities.

The ISSEP model focuses strictly on the development phase of the system and software engineering life cycle and the interfaces between systems and software. The development phase of the life cycle defines the parts of the system to be developed and the processes for developing, implementing, and using each part. The model does not include activities from other parts of the life cycle, such as deployment or disposal. Future versions of the ISSEP model may address other parts of the life cycle.

1.2 OBJECTIVES

The main purpose of the ISSEP model is to enable improvement of the overall systems development process by allowing systems and software engineers to perform their work more efficiently. To achieve this goal, ISSEP deals with the complexity associated with developing large, multifaceted systems by providing mechanisms for coordinating and controlling the development effort. The ISSEP model takes a holistic view of systems development because many of the coordination and control challenges take place where systems, software, and management interface. The objectives of the ISSEP model are to:

- Provide a balanced integration among the systems, software, and management activities
- Support a broad range of applications and projects
- Comply with major systems and software engineering standards

When integrating the systems, software, and management activities into the total model, ISSEP strives to balance their interplay so that no discipline is over emphasized at the expense of another. The ISSEP

model defines the critical information flows between activities. The model also defines its interfaces by identifying where this information is created and describing where and how it is used.

The ISSEP model's scalability ensures that the process is appropriate for use on any size project. The model is scalable because it defines a generic process that is applicable to the development of systems and software regardless of size. The ISSEP model defines the process for developing the parts of the system (including software parts) and provides a process framework for integrating them into the complete system.

The ISSEP model was developed by investigating and adapting existing processes and process frameworks. Building on well-known and accepted standards establishes a solid foundation. To help validate the ISSEP model, the Consortium had the model reviewed by external reviewers, including individuals involved with the international and national systems and software process standards and representatives from industry.

1.3 INTENDED AUDIENCE

The primary audience for this technical report consists of those professionals responsible for creating or modifying a systems and software engineering process. These professionals are usually the process developers (process engineers) at the organizational or project level (i.e., those who define the process to be used by the organization as an organizational standard process or instantiate it for use by a project). The secondary audience is systems and software engineers. Their interest is to gain knowledge and buy-in to the process. This report assumes the reader has a familiarity with the applicable system and software standards listed in Section 2.

The ISSEP model focuses on the process information flow and identification of critical systems and software engineering process interfaces. The ISSEP model provides process developers and process improvement engineers with an understanding of the information and activities that are critical for ensuring a smooth transition between systems and software engineering. The process provides guidance for project practitioners, including managers, developers, and specialty disciplines, by defining where information is created, how it can be communicated, and where the information is needed for effective decision making.

1.4 ORGANIZATION

This report includes the following sections and appendixes:

- **Section 1, Introduction.** This section describes the ISSEP objectives, defines the primary audience for the report, and provides an overview of each report section.
- **Section 2, Process Issues.** This section describes the important issues that ISSEP was developed to address.
- **Section 3, The ISSEP Model.** This section explains the ISSEP model structure and describes the model in detail.
- **Section 4, ISSEP Concepts and Rationale.** This section describes the ISSEP model's systems and software interface, information flow, development plan, and risk management concepts.

- **Section 5, Applying the ISSEP Model.** This section discusses how to apply the ISSEP model. It describes process tailoring, how to get started applying ISSEP to a development effort, and how to use the ISSEP model with various life-cycle models.

- **Section 6, Process Issues Revisited.** This section describes how the ISSEP model addresses the process issues introduced in Section 2.

- **Appendix A, ISSEP IDEF0 Diagrams.** This appendix contains the IDEF0 diagrams of the ISSEP model.

- **Appendix B, ISSEP Activity Descriptions**. This appendix contains the ISSEP model activity descriptions in alphabetical order.

- **Appendix C, ISSEP Information Flow Descriptions**. This appendix contains the ISSEP model information flow descriptions in alphabetical order.

- **Appendix D, Develop Operational System Context (A-1).** This appendix contains a description of an example context for development of the operational system defined by the ISSEP model.

- **Appendix E, Tool Support Environment**. This appendix describes the tool used to define the ISSEP model and available formats for the ISSEP model.

- **Appendix F, Mapping to Standards**. This appendix contains a mapping of the activities in MIL-STD-498 to the ISSEP model activities and the mapping of the Capability Maturity Model's key process areas to the ISSEP model activities.

1.5 USING THIS REPORT

Appendixes A, B, and C contain the ISSEP model definition. Appendix A contains the IDEF0 model diagrams; Appendix B contains descriptions for each of the activities in the IDEF0 diagrams; and Appendix C contains descriptions for each of the information flows in the IDEF0 diagrams. The reader should consult these appendixes for more specific information about the ISSEP model when reading this report.

Sections 1 and 2 provide background and rationale for the ISSEP model's creation. Section 3 provides a textual description of the model and includes a high-level model description that explains the basic model decomposition strategy and a detailed description of each of the IDEF0 diagrams. The purpose of Section 3 is to provide the reader with a detailed understanding of the model's structure and contents and the flow of information through the model.

Section 4 describes the important ISSEP model concepts and rationale for how they are modeled. Section 5 describes how to apply the ISSEP model and provides tailoring guidance, explanations of how to begin applying the ISSEP model on new and ongoing projects, and a brief example describing an application of the ISSEP model. Section 6 revisits the process issues defined in Section 2 and explains how the ISSEP model addresses each issue. Sections 4, 5, and 6 refer to the ISSEP model as described in Section 3 and assume a familiarity with it.

1.6 TYPOGRAPHIC CONVENTIONS

This report uses the following typographic conventions:

Serif font ..... General presentation of information.

Italicized serif font ..... ISSEP information flow names and publication titles.

Boldfaced serif font ..... ISSEP activity names and section headings.

Boldfaced italicized serif font ..... Run-in headings in bulleted lists and low-level titles.

2. PROCESS ISSUES

2.1 OVERVIEW

This section describes the process issues considered when developing the ISSEP model. The ISSEP model was created to meet the following objectives (listed in order of decreasing priority):

- Define an integrated set of management and technical activities for developing systems containing software components
  - Define the interfaces between the management and technical activities
  - Define the interfaces among management activities
  - Define the interfaces among technical activities
  - Define the interfaces between the systems and software development processes
- Ensure compliance with existing standards and process frameworks
- Manage complexity when developing large, multifaceted systems
- Provide process adaptability and tailorability so that a broad range of applications and project environments can be addressed while accommodating each project's unique characteristics

The following sections provide additional detail on each objective. Section 6 revisits these process issues and describes the ISSEP model's approach to addressing each.

2.2 INTEGRATED MANAGEMENT AND TECHNICAL ACTIVITIES

ISSEP provides a balanced process that equally emphasizes the management and technical perspectives. Successful management requires technical insight, and successful technical activities must be well managed.

The management activities provide the control necessary for developing a system. These activities focus on the plans needed to develop the system (e.g., resources, budgets, schedules) and the performance of the technical activities relative to these plans. Management activities are also responsible for making midcourse corrections to the plans to ensure successful system development.

The technical activities define both the systems and software engineering activities necessary to develop systems containing software. Specifically, these activities include:

- Analyzing and specifying system requirements
- Allocating requirements to software, hardware, and people
- Analyzing and specifying software requirements
- Designing components
- Verifying and validating components
- Integrating components

The management and technical activities work together to ensure that the development effort has clear development goals based on customer needs. The management activities strive to establish a rapport with the customer which aids in the creation of the operational concept which directs the development effort.

The balanced approach to the integration of management and technical activities ensures that neither set of activities constrains the other or is overemphasized. The ISSEP model's integration addresses the issues of what information interfaces exist between activities, how and where this information is created, and how and where this information is used.

The information flow in the ISSEP model focuses on the:

- Information required to perform an activity
- Process interfaces between critical systems and software engineering activities
- Feedback of information used for effective decision making

The ISSEP model defines the minimum set of required interfaces between management and technical activities, among the management activities, and among the technical activities. These interfaces provide either the necessary information to perform a process activity or provide feedback information necessary to identify and mitigate risk. Both management and technical information is needed for effective implementation of an activity. The management information provides the control mechanisms, (e.g., schedule, resources assigned, budget constraints). The technical information provides the system descriptions and/or products (e.g., requirements, hardware/software allocation, design architecture) needed to perform an activity, as well as report status and risks to the management activities.

The interfaces between the ISSEP model process activities define the coordination and communication mechanisms that ensure successful system/software deliveries. The ISSEP model focuses on those interfaces that are critical and could result in excessive rework if decisions are made without adequate information. For example, integrating hardware and software components can cause rework if component interfaces are not adequately defined.

Finally, some information flow from an activity contains requests for action. The responding activities inform the originating activity(ies) of the action taken and its results. This type of information flow is

necessary to ensure that information reaches its appropriate destination and has the anticipated results. These feedback loops can originate with or terminate at both the management and technical activities.

The ISSEP model takes a high-level, engineering view of the development process, which includes explicit management and technical activities. There are a number of other views that could have been used to define the ISSEP model such as a configuration management view and quality assurance view. Although interesting, these views are not explicitly included in this version of the ISSEP model.

2.3 STANDARDS COMPLIANCE

The ISSEP model was created to include, extend, and leverage the work of others as much as possible. National and international standards heavily influenced the development of ISSEP. These standards include MIL-STD-498 (Department of Defense 1994), EIA/IS-632 (EIA 1994), P1220 (IEEE 1994), ISO/IEC 12207 (ISO/IEC 1995), the Systems Engineering Capability Maturity Model℠ (SE-CMM) (Software Engineering Institute 1994), and the Capability Maturity Model (CMM℠) for Software (Paulk et al. 1993). The following list gives a brief summary of these inputs:

- **MIL-STD-498: Software Development and Documentation.** This interim military standard focuses on establishing requirements for any type of software development and developing the associated documentation. The standard identifies 19 activities covering the development life cycle, from project planning to system and software development and deployment. This standard references Data Item Descriptions that define the contents of the required documentation for each life-cycle activity.

- **EIA/IS-632: Systems Engineering.** This interim commercial standard is based on MIL-STD-499B. It identifies and defines the systems engineering tasks to be performed iteratively throughout the system life cycle and describes the use of a Systems Engineering Management Plan for defining and controlling an integrated product and process development program.

- **P1220: Standard for Application and Management of the Systems Engineering Process.** This trial-use standard describes eight elements of a systems engineering process and how the process should be applied at each of six stages of a typical system life cycle, from system definition to customer support. A National Standard for Systems Engineering is expected to result from the merging of this standard with EIA/IS-632.

- **ISO/IEC 12207: Information Technology—Software Life Cycle Processes.** This international standard is currently being adapted to create ANSI 016, the future U.S. commercial standard for software development, which will also include the technical content of MIL-STD-498. ISO/IEC 12207 identifies 17 processes in the life cycle of software, divides each of them into a set of activities, and divides the activities into sets of tasks. The scope of this standard is the acquisition, supply, development, operation, and maintenance of software products.

- **Systems Engineering Capability Maturity Model (SE-CMM).** The SE-CMM describes the essential elements of an organization's systems engineering process. It does not specify a process, but rather provides a set of criteria that can be used when evaluating systems engineering practices.

- **The Capability Maturity Model (CMM) for Software.** The CMM describes the key goals and practices necessary for consistent quality in developed software. Like the SE-CMM, it provides a set of evaluation criteria that can be used for assessing the maturity of various elements of a software development process (e.g., software configuration management, software project planning, peer reviews, intergroup coordination).

These are the emerging standards, both commercial and military, with the most direct impact on systems and software development. It is important that processes, like the ISSEP model, comply with these standards in order to meet a minimal set of accepted government and industry best practices.

In general, these standards and processes describe required systems and software development activities and may additionally describe the work products produced by the activities. The ISSEP model provides a high-level process framework for implementing these standards, by defining activities and information flows for integrating the requirements documented in the standards. Thus, the ISSEP model provides guidance not found in these standards and complements them by including important interface descriptions. Although compliant with all of the above standards, the ISSEP model does not include all the activities defined in the standards. The activities that are not explicitly contained in the ISSEP model are either not within the scope of the model or not explicitly addressed in the model's high-level management and development view.

2.4 MANAGING COMPLEXITY

An important consideration when developing systems is the ability to manage complexity and facilitate engineering of large, multifaceted systems. The ISSEP model addresses managing complexity with recursion and increments. The ISSEP model assumes that each system contains a hierarchical set of system parts (e.g., software subsystems, hardware subsystems). The model is applied recursively for each system part, and each part is responsible for communicating information up to its parent. The model defines an integrated process for design, development, and integration of the parts by defining the information and coordination needed by higher and lower levels of the decomposed system.

The ISSEP model takes an evolutionary perspective in aiding complexity management. In other words, the ISSEP management activities divide the development objectives for the system into manageable subsets called increments. Defining increments results in determining the order in which technical activities are completed. Examples of activities that define an increment include producing particular work products (e.g., a requirements specification), producing a prototype, and mitigating a critical risk.

2.5 PROCESS ADAPTABILITY AND TAILORABILITY

The ISSEP model is a generalized development process; therefore, it is adaptable to a broad range of applications and project environments. Although the process model is defined at a high level, a very detailed, enactable process can be derived by tailoring the process. Once tailored, the ISSEP process is instantiated (i.e., specific resources, methods, and tools are assigned), and the instantiated process is executed.

The benefits associated with an organization adopting a generalized model that can be tailored for specific applications include familiarity with the process, continuous process improvement, and organizational process standardization. Because the same process is used repeatedly, practitioners

become familiar with the process and need not be retrained for each project. Processes do not need to be reengineered or recreated for each project. As knowledge is gained from applying the process, it can be used to improve the process. As an organization's understanding of the process grows, the process can be customized to address the organizational needs and goals in an efficient manner.

Process tailoring is defined as creating a specific process from a general one. To tailor the ISSEP model, a project must consider its unique process drivers. A process driver is a characteristic of the project that influences the definition of the detailed process activities. Process drivers include characteristics such as the development strategy (e.g., waterfall, iterative, evolutionary), cost model or contractual constraints, imposed standards, and required milestones. Before the process can be enacted, each activity must be instantiated by choosing the methods, tools, techniques, and resources for performing the activity.

Tailorability is critical for generic process models. If process models are too inflexible, they complicate tailoring by forcing reengineering of the process (i.e., adding or eliminating activities or information flow from the model) or inhibit the specification required to make a process enactable. As discussed in Sections 5.1 through 5.3, process tailoring plays an important role in creation of realistic and enactable plans.

This page intentionally left blank.

3. THE ISSEP MODEL

This section describes the ISSEP model, including high-level activity and information flow descriptions, and the modeling approach and notation used. Section 4 describes the concepts and rationale behind the ISSEP model. Appendixes A, B, and C contain the ISSEP model's IDEF0 diagrams, detailed activity descriptions, and detailed information flow descriptions, respectively.

3.1 ISSEP STRUCTURE

Systems, especially large systems, are decomposed into smaller parts to simplify development. Breaking a system into smaller parts makes the problem easier to solve and the parts easier to develop. Each part can be developed independently, sometimes in parallel, and the parts can be integrated to produce the system. Although this strategy makes each part easier to develop, integrating the parts can produce problems that may be difficult to solve. Many of the problems with development of large, complex systems are, in fact, associated with integration. The solution is to develop a means of decomposing the system into parts that can be independently developed and integrated with relative ease to produce the complete system.

There are two important aspects to the solution: ensuring that the sum of the decomposed parts is sufficient for completion of the system (i.e., no significant pieces are missing), and ensuring the parts can be integrated once produced. Many current development methods aid in decomposing the system (e.g., object-oriented methods). Development methods are necessary, but in the end it is unrealistic to assume that any method, no matter how rigorously applied, provides flawless results. All system and software development efforts require a process framework that provides the ability to recognize potential development problems and take corrective actions early, before problems become more costly to solve.

Therefore, the ISSEP model defines the decomposition strategy for system development, which includes information exchange and risk management mechanisms so that potential problems are addressed early in the life cycle. The ISSEP model focuses on planning for the integration in design and implementation at all levels of the decomposed system. The following sections discuss how the ISSEP model addresses these problems.

3.1.1 DECOMPOSITION STRATEGY

A system decomposition is a hierarchy of system parts (see Figure 1). The total, integrated system is at Level 0, or the root level. In the simplest case, customer needs (e.g., system requirements) are input to the root level, and the completed system is output. If the system were small and simple, it could be implemented at the root level (Level 0), and there would be no need for either decomposition or

integration. The decision about whether the system needs to be decomposed and, if so, a description of each decomposed part, is made when the system is designed and the development risks are analyzed.

System parts are defined as the results of the decomposition. In Figure 1, the system parts are segments, subsystems, software and hardware configuration items (CSCI and HWCI), hardware components, and software units. Figure 1 illustrates a sample decomposition in which Level 0 is the system level, Level 1 is the segment level, Level 2 is the subsystem level, Level 3 is the configuration item (CI) level, Level 4 is the component and unit level, and Level 5 is the unit level. Some of the parts in Figure 1 have not been decomposed into components so that the entire figure could be scaled to fit the page width.

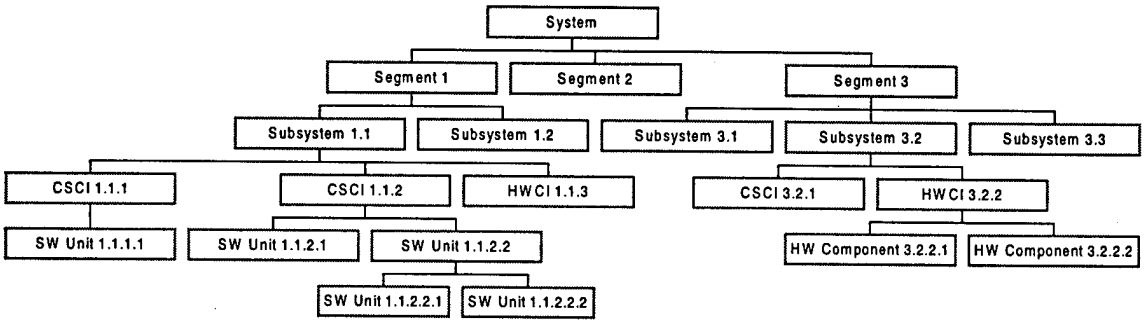


Figure 1. System Hierarchy

A design activity exists at each level in the hierarchy. The design activity at each level defines the:

- Decomposition at the next lower level, if needed
- Requirements for each of the decomposed parts
- Risks associated with satisfying the requirements passed down from the parent level
- Risks associated with the lower-level decomposition that have been passed up from below

The design activity at the higher level defines the system decomposition for the next lower level and passes its development requirements to each child. The child design activity analyzes the requirements and defines its development strategy accordingly. The parent design activity receives each child's decomposition strategy, implementation plan, and development strategy. The parent design activity is then responsible for determining whether development should continue as defined by each child. If changes to the parent or child decomposition are necessary, the design activities in the new or redefined system parts perform the same actions, and so on until an acceptable decomposition is defined. As the system is further and further decomposed, the decomposition strategy, the implementation plan, and the risks associated with each level of decomposition are passed up the hierarchy for analysis.

If the system contains software, at some point in system decomposition, software-only system parts are defined. The ISSEP model is not restricted to development of systems with software components; however, the focus for this version of the model is software-intensive systems. Although the details of the design activity in the ISSEP model differ for systems and software parts, the decomposition strategy is the same. Software parts are decomposed, and their design activities pass risk and design information back to the system part from which they were decomposed.

The ISSEP model is intended to define the design process at each level of the system hierarchy. Therefore, the ISSEP model is instantiated hierarchically to match the system decomposition. That is, ISSEP is used at each level of the system decomposition to aid in developing the next lower-level decomposition.

The ISSEP model can contain as many process levels as appropriate for the system under development. However, the ISSEP model has divided the different system parts identified in Figure 1 into three groups based on the process used to implement that part. The groups are system, CI, and component. The ISSEP model defines the system grouping to include the system, segment, and subsystem parts. The process for developing these parts includes the system engineering design activities. The ISSEP model defines the CI grouping to include software and hardware CIs and software units that are further decomposable. The process for developing these parts includes the software and hardware design activities, but does not include the implementation of the software or hardware. The ISSEP model defines the actual implementation of software and hardware in the process for developing components. Components, therefore, cannot be decomposed and represent the leaves of the hierarchy.

In the most basic case, the ISSEP model consists of two levels of decomposition: the system level and the CI level. In this case, the higher level is the system level and is named the Operational System level and the lower level is the CI level (see Figure 2). A distinction is made between the two levels because the ISSEP model defines the system and CI development processes differently. However, these two levels are decomposable and can create a process with as many levels as necessary to develop the system.

Figure 2 shows three boxes outside the dashed boxes—**Manage System Development**, **Design and Verify System**, and **Integrate and Test System**—that define the Operational System level. The four boxes within the dashed boxes—**Manage CI Development**, **Design and Verify CI**, **Develop Component**, and **Integrate and Test CI**—define the activities for each Configuration Item level. The operational system can be decomposed into more than one CI, as noted with the dashed boxes shadowing the dashed box in the foreground. The CIs, in turn, can be decomposed into more than one component. The solid boxes that shadow the **Develop Component** box similarly represent the components that makeup the CI.

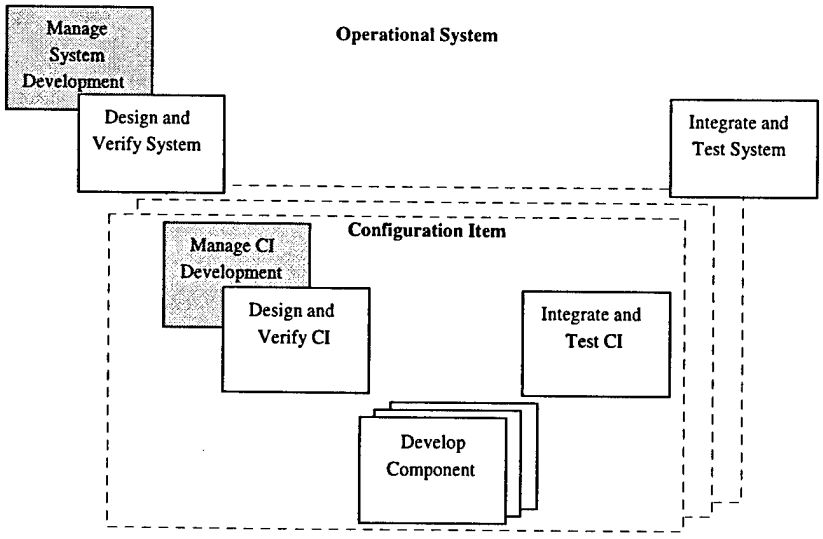


Figure 2. ISSEP Model Decomposition

Figure 2 shows only the activities that need to be performed for each system hierarchy level. For simplicity, the interfaces and information flow between activities are omitted in this figure. However, the information flows are the vital link to ensure successful integration between levels and, ultimately, a successful system.

3.1.2 THE INTEGRATION EMPHASIS

Most of the activities traditionally associated with integration and testing take place in the integrate and test activity boxes (e.g., **Integrate and Test System** and **Integrate and Test CI**) included at every decomposition level. However, in the ISSEP model, integration considerations begin during the design. The design activities are named **Design and Verify System** and **Design and Verify CI** to reflect the fact that they encompass more than the traditional design activities.

The management boxes in Figure 2 have a gray background for emphasis. Every level has an associated management box representing that level's planning and tracking activities. The management activities control the flow of information. These activities receive information from above, below, and within the process hierarchy. The information is used in producing plans that direct the development effort, including integration analysis and planning.

During design, when the decomposition is being analyzed, one of the critical evaluation factors is integration risk. Integration risks are identified and documented as part of the design along with the integration and test procedures and integration plan. These procedures and plan are the by-products of careful integration analysis done during design and used by the management activities during planning. Thus, in much the same way that the decomposition strategy, at each decomposition level, is evaluated based on design and implementation risks, it is also evaluated based on integration risks.

The remainder of this section describes the full ISSEP model, including activity and information flow descriptions, emphasizing decomposition and integration.

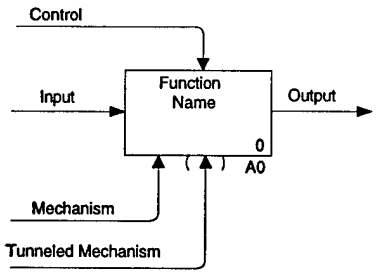
3.2 ISSEP NOTATION AND MODEL STRUCTURE

This section describes the notation used to model ISSEP and provides the hierarchy of the ISSEP model.

3.2.1 IDEF0 NOTATION

The IDEF0 notation was selected for modeling ISSEP because it is widely used for "developing structured graphical representations of a system or enterprise" (Department of Commerce 1993). The tool used for generating the IDEF0 diagrams and reports is System Architect/Business Process Reengineering (SA/BPR) (Popkin Software and Systems, Incorporated 1991–95).

The model is consistent with the IDEF0 standard (see Figure 3), so that a "box" is a rectangle representing a function, and an "arrow" is a directed line representing the movement of data or objects. Each box on a diagram has a number in the bottom right corner to identify it within the diagram. If the box refers to a child diagram representing the decomposition of the function, then the number of that child diagram is placed below the bottom right corner of the box. "Input arrows" enter the left side of a box, and "output arrows" leave from the right side. "Control arrows" enter the top of the box, and "mechanism arrows" enter the bottom of a box.



Input arrows represent data or objects required by a function to produce the desired outputs. Output arrows represent what is produced by the function. Control arrows specify the conditions required for the function to produce correct outputs. Input and control arrows differ in that control arrows provide guidance that determines how the inputs are to be used in creating the outputs. A mechanism arrow represents the means used to perform a function.

Figure 3. IDEF0 Activity

In the ISSEP model, a plan is the usual data in a control arrow because the plan allocates schedule and budget and defines the scope of work to be performed by the activities. Without the plan, an activity is unconstrained; therefore, the activity may fail to meet important time and cost considerations and may not produce all aspects of the desired output (e.g., the output may not have the desired scope, structure, or content). For example, the system design plan constrains the design activity. The input arrows to the design activity include the system requirements, which are to be transformed by the activity into a system design. In ISSEP, the important mechanism is the *Development Environment.*

Arrows (and their meanings) may be combined through "bundling" and separated through "unbundling," as represented by joins and forks. Hierarchical decomposition of functions is represented by "child diagrams" that show the detail within each decomposed box. The same arrows enter and leave a child diagram as enter and leave the parent box. However, if an arrow enters a box, but is not shown entering the corresponding child diagram, then its arrowhead is marked with parentheses. This is known as tunneling. For the ISSEP model, the data or objects represented by a tunneled arrow enter every box in the child diagram.

3.2.2 Model Structure

Table 1 lists the hierarchy of diagrams representing the ISSEP model and gives the corresponding figure numbers for the diagrams in this section and the enlarged versions in Appendix A. Diagram A-0, Context, represents the ISSEP model as a single box. Diagram A0, Develop Operational System, is the top level decomposition of the process and will be used to introduce some of the important features of the model.

Table 1. Diagram Structure for the ISSEP Model

Diagram Number	Diagram Title	Section 3 Figure Number	Appendix A Figure Number
A-0	Context	Figure 4	Figure 16
A0	Develop Operational System	Figure 5	Figure 17
A1	Manage System Development	Figure 6	Figure 18
A2	Design and Verify System	Figure 7	Figure 19
A3	Develop Configuration Item	Figure 8	Figure 20
A31	Manage CI Development	Figure 9	Figure 21
A32	Design and Verify CI	Figure 10	Figure 22
A33	Develop Component	Figure 11	Figure 23
A34	Integrate and Test CI	Figure 12	Figure 24
A4	Integrate and Test System	Figure 13	Figure 25

Diagrams A0 through A4 are the focus of this technical report. Diagram A-0 provides the external interfaces to the Operational System.

3.3 ISSEP DEFINITION

This section focuses on diagrams listed in Table 1. Each subsection contains a diagram and a description of the activities and information flows. Diagram A-1 illustrates a typical context for the system development and resides in Appendix D (see Figure 26). The layout of the activities in the IDEF0 diagrams in the ISSEP model does not represent time sequencing or dependencies; rather, the layout defines the information flows between activities. The objective of the layout design is to make information flows as clear and easy to follow as possible.

In the descriptions that follow, occasionally there are references to information flows that do not appear on the associated diagram. These information flows are either bundled into the flows that are on the diagram (see information flow descriptions in Appendix C) or are flows that are on the referenced child diagram (see the child diagram and description for additional information).

3.3.1 CONTEXT (A-0)

Figure 4 defines the external interfaces for the **Develop Operational System** activity.

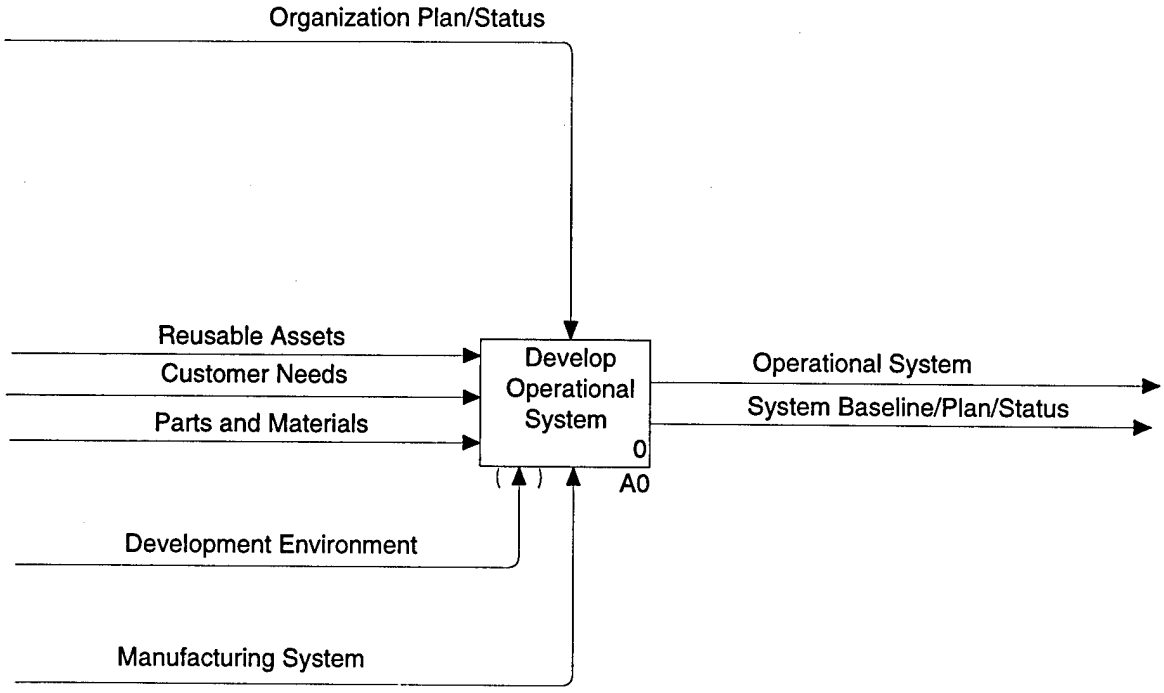


Figure 4. Context (A-0)

The control *Organization Plan/Status* contains the planning documents and the associated status information used to guide and constrain the system development, such as organization structure and objectives, cost and resource constraints, and organizational policies and procedures. The plan and associated status provide context for managing the system development.

Reusable Assets consist of developed system parts and their associated documentation, such as requirements, design and design rationale, integration and test plans, unit test cases, results of testing, certification documentation, user documentation, and management plans and status information. These reusable assets are available for inclusion in the developing system, as needed. *Customer Needs* define the customers' and stakeholders' goals for the system from its conception until it is decommissioned, including the reasons for the system's existence. In addition, the *Customer Needs* define the operational concept that describes how the system is intended to function, the measures of system effectiveness, critical influencing factors, customer requirements, and customer expectations. *Parts and Materials* refer to the hardware items that are used in the creation of hardware components.

The mechanism *Development Environment* consists of the tools, methods, and people that will execute the development process. The mechanism *Manufacturing System* supports the fabrication of hardware components and the integration and test activities.

The *Operational System* that is produced is the system as delivered to the customer and may include maintenance and user documentation. The *System Baseline/Plan/Status* includes planning documents and associated status information, design documentation, integration and test information, and software source code, which meet the objectives of the entire project.

3.3.2 DEVELOP OPERATIONAL SYSTEM (A0)

Figure 5 defines the system-level activities and is a decomposition of the Context (A-0), and therefore, has the same external interfaces. There are three system-level activities and one configuration item-level activity. One of the system-level activities, **Manage System Development**, manages the two technical activities, which perform the design, verification, integration and testing of the system. **Manage System Development** also provides overall control to the configuration-item-level activity. The configuration-item-level activity includes the management and technical subactivities required to develop a CI.

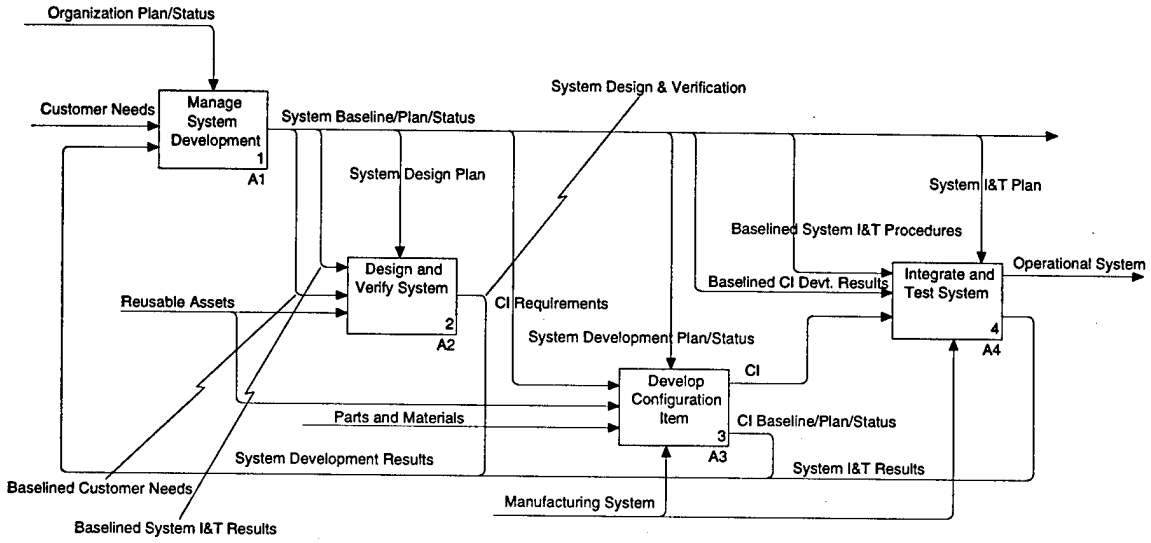


Figure 5. Develop Operational System (A0)

The **Develop Operational System** activity in the decomposition contains the following activities:

- Box 1, **Manage System Development,** plans, controls, and coordinates the development of the system. This activity uses the *Organization Plan/Status* and the *Customer Needs* as a basis for planning the system development, including the definition of the system increments. The results of the design, development, and integration and test activities, along with status information gathered from the *System Development Results,* which includes the *System Design & Verification,* the *System I&T Results,* and the *CI Baseline/Plan/Status,* are used to produce the *System Baseline/Plan/Status,* which grows as system development progresses to include all nontangible parts of the system.

- Box 2, **Design and Verify System,** evolves a *System Design* from the *Baselined Customer Needs* by analyzing those needs to define the *System Requirements.* The optimum design solution is selected from alternative functional and physical architectures. This activity uses the adaptable system requirements and functional and physical architectures contained in the *Reusable Assets* if they can be adapted to the system under development. The *System I&T Procedures* are produced for later use by the **Integrate and Test System** activity. After integration and test is complete, the *Baselined System I&T Results* are input back into this activity so that the results can be verified and it can be determined whether the system is ready for delivery to the customer.

- Box 4, **Integrate and Test System,** assembles and tests the hardware and software configuration items according to the *Baselined System I&T Procedures*, which include test cases and expected results, and documents the outcome in the *System I&T Results*, which describe the status of the integration and test process and any exceptions to the observations expected by the procedures. This activity produces the *Operational System,* which is delivered to the customer. The *Operational System* is the tangible part of the system, but it may contain nontangible items such as software and user documentation.

The following bullet describes the configuration-item-level activity:

- Box 3, **Develop Configuration Item,** creates an integrated and successfully tested configuration item that meets the *CI Requirements* generated in the **Design and Verify System** activity and baselined in the **Manage System Development** activity as part of the *System Baseline/Plan/Status*. This activity uses the *System Development Plan/Status* as a control and documents any status information and risks as part of the *CI Baseline/Plan/Status*. In the case of a hardware CI, the input *Parts and Materials* and the mechanism *Manufacturing System* are used to create any tangible components. This activity also uses the adaptable CI requirements and architectural and detailed designs contained in the *Reusable Assets* if they can be adapted to the CI under development.

3.3.3 MANAGE SYSTEM DEVELOPMENT (A1)

Figure 6 shows the ISSEP management activities at the system level. This activity is responsible for maintaining baselines of the system-level products and managing the system-level activities. This activity defines the system context (e.g., objectives, goals, stakeholders), analyzes system risks, and uses this information along with the current development status to produce the system development plan. The management activities are repeated every increment; thus, the system context, risks, and plan are reviewed, and the plan is modified to reflect the current status and is updated, as necessary, to ensure that the project remains focused on critical project objectives.

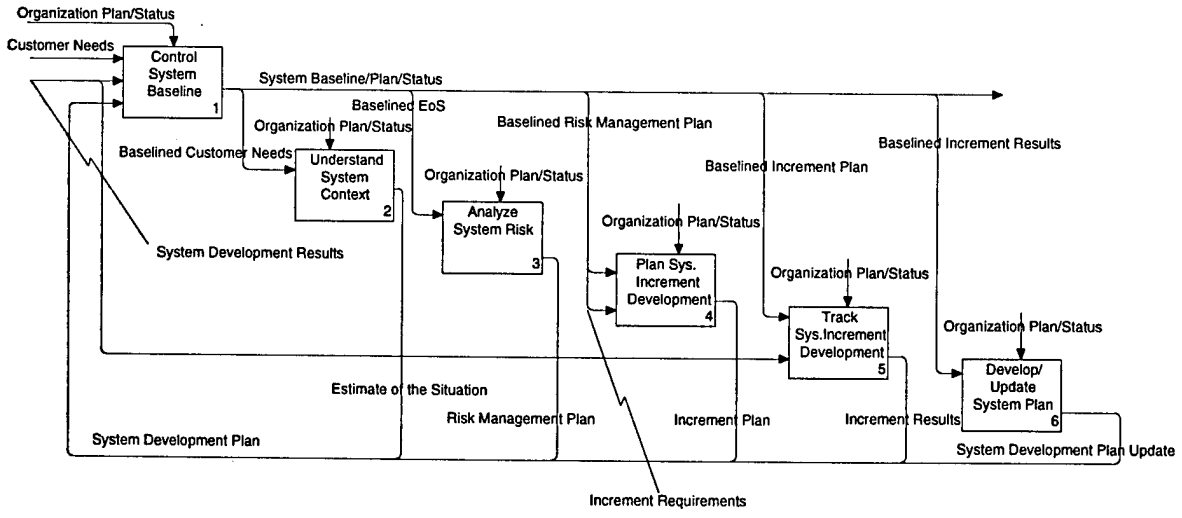


Figure 6. Manage System Development (A1)

The following list describes the management activities:

- Box 1, **Control System Baseline**, establishes a product baseline for the system. Every nontangible part of the system, at some point, is input into this activity. Each part is reviewed to determine whether it qualifies for baselining and, if accepted, is added to the current system baseline that is output. Even the outputs from the other management activities are baselined in this activity. The rigor of the review and baseline functions are determined by the system development plan and can range from an informal review and creation of a new version of the system baseline to a formal acceptance followed by formal configuration management. As part of the review activity, changes to the previous baseline are noted, and these changes are stored, tracked, and analyzed as part of the planning process. This activity serves as a synchronization point between the technical baseline and management plans. Every time the baseline is modified, the other management activities in this diagram are performed. This procedure ensures that the current plans always reflect the current baseline and vice versa.

- Box 2, **Understand System Context**, identifies factors that could have an influence on the success of the system development and defines the scope of this increment of the development. The *Baselined Customer Needs* are analyzed, along with other pertinent parts of the current *System Baseline/Plan/Status*, to determine the increment objectives and constraints and to identify alternatives for meeting the objectives while remaining within the constraints. This information is documented in the *Estimate of the Situation.*

- Box 3, **Analyze System Risk**, identifies potential increment risks and analyzes the risks to determine which are critical to the development effort and when mitigation action is recommended. The activity uses this information to develop a set of mitigation strategies for each risk and a time table for implementing the mitigation strategies. The main source used for identifying the risks is the *Estimate of the Situation,* and the output from the analysis is the *Risk Management Plan.*

- Box 4, **Plan System Increment Development**, creates the detailed development plan for the next increment. This activity uses the *Baselined Risk Management Plan* and the *Increment Requirements* to determine how to achieve the increment objectives and mitigate risk. Development goals for the increment are established and used as a basis for selecting a development strategy. Detailed size, cost, and schedule estimates are made. The development process for the increment is tailored and instantiated, and detailed work assignments are documented. This detailed planning remains within the scope of the *System Development Plan*, adding detail, as necessary, to make the plan enactable. The output of this activity is the *Increment Plan.*

- Box 5, **Track System Increment Development**, uses the *System Development Results* to assess the progress and analyze the seriousness of situations that arise during development. This activity controls the enactment of the *Baselined Increment Plan* and ensures that the development is enacted according to plan. When the development deviates too far from the plan or when the development goals documented in the plan are met, this activity terminates the development activities and initiates the **Develop/Update System Plan** activity.

- Box 6, **Develop/Update System Plan**, uses the *Baselined Increment Results* to create the *System Development Plan.* This plan defines each of the development increments at a high level. If a

System Development Plan already exists, this activity updates the plan based on the results of the past increment development efforts, including lessons learned, newly identified risks, and status information. If this is the first increment, this activity generates the first version of this plan from the context information and risk analysis. This plan is a living document and is kept accurate and current.

3.3.4 DESIGN AND VERIFY SYSTEM (A2)

Figure 7 defines the system design and verification activities in the ISSEP model. The major input is the *Baselined Customer Needs*, and the major control flow is the *System Design Plan.* There are three design activities that are followed by two evaluation and verification activities. The results of the evaluation, validation, and verification are fed back and used to make appropriate modifications, and the revisions are also evaluated, validated, and verified.

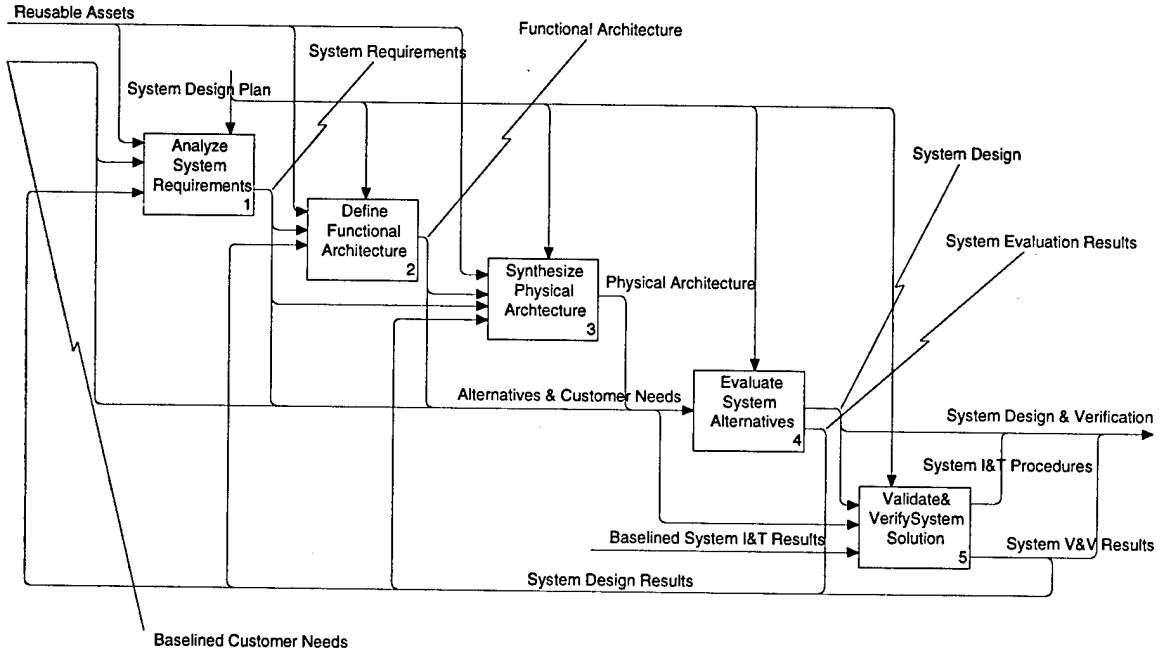


Figure 7. Design and Verify System (A2)

The following list describes the design activities:

- Box 1, **Analyze System Requirements,** examines the *Baselined Customer Needs* to assess the problems the system is to solve, determines the needs that the system is to address, defines the environment in which the system is to operate, and defines the requirements that the system must satisfy to be acceptable to the user and customer of the system. This activity uses the adaptable system requirements contained in the *Reusable Assets* if they can be adapted to the system under development. The resulting *System Requirements* will then define the behavioral and performance requirements for the system that, when met, satisfy the system developer's obligations in the production of the system.

- Box 2, **Define Functional Architecture,** creates a *Functional Architecture* by partitioning the *System Requirements*. The *Functional Architecture* is made up of a hierarchy of functions, their internal behavior, and their interfaces. These interfaces can be electrical, mechanical, or logical. Interfaces define the interactions of the functions with each other as well as with the external environment. Using criteria that include performance and design considerations, this activity identifies alternative feasible solutions that meet the requirements. The adaptable functional architectures contained in the *Reusable Assets* are used if they can be adapted to the system under development.

- Box 3, **Synthesize Physical Architecture,** allocates the *System Requirements* and the elements of the *Functional Architecture* to a *Physical Architecture* that defines the viable alternatives in terms of hardware, software, and people (procedures). This activity defines where the functions are accomplished, the technical parameters that drive the performance of the parts of the system, and how the interfaces communicate the interactions among the parts. The adaptable physical architectures contained in the *Reusable Assets* are used if they can be adapted to the system under development. This activity identifies alternative feasible solutions that implement the requirements and functions.

The following list describes the evaluation and verification activities:

- Box 4, **Evaluate System Alternatives,** performs trade studies of the alternative functional architectures to select the alternative that best supports the *Baselined Customer Needs* and *System Requirements*. The physical solution alternatives are analyzed to determine which one best satisfies the allocated functional and performance requirements, interface requirements, and design constraints. The resulting *System Design* identifies the preferred alternative based on a comparison of all alternatives and includes a record of requirements, alternatives, and design decisions. The *System Evaluation Results* document the studies and any proposed improvements.

- Box 5, **Validate & Verify System Solution,** evaluates the *System Requirements* to ensure that they represent the *Baselined Customer Needs* and project constraints and that all operations and support concepts have been fully addressed. The completeness of the *Functional Architecture* is assessed to determine whether the validated requirements are satisfied. The *Functional Architecture* verifies that the *Physical Architecture* is traceable to the verified *Functional Architecture* and to the validated *System Requirements*. The *System I&T Procedures* describe how the hardware and software CIs are to be progressively assembled and tested to determine compliance with the *System Requirements* and *System Design*. Analyzing the *Baselined System I&T Results,* which document the outcome of the **Integrate and Test System** activity, determines whether any changes have to be made to the requirements or architecture of the system and verifies that the system is complete and ready for delivery. The *System V&V Results* document the results of any form of verification and/or validation completed on any of the work products produced in the design of the *Operational System*, including testing the system itself.

3.3.5 Develop Configuration Item (A3)

Figure 8 defines the activities necessary to develop a single CI. Although there is only one **Develop Operational System** activity for a given operational system, there may be multiple **Develop Configuration Item** activities. For each instance of this activity, there is a **Manage CI Development** activity to manage development of the entire CI and a **Design and Verify CI** activity that elaborates

requirements and produces an architectural and detailed design. There will be as many **Develop Component** activities as necessary to code and unit test (or in the case of reusable components, to procure) the components. Finally, there is a single **Integrate and Test CI** activity to assemble cohesive collections of components into builds (ultimately, into the entire CI).

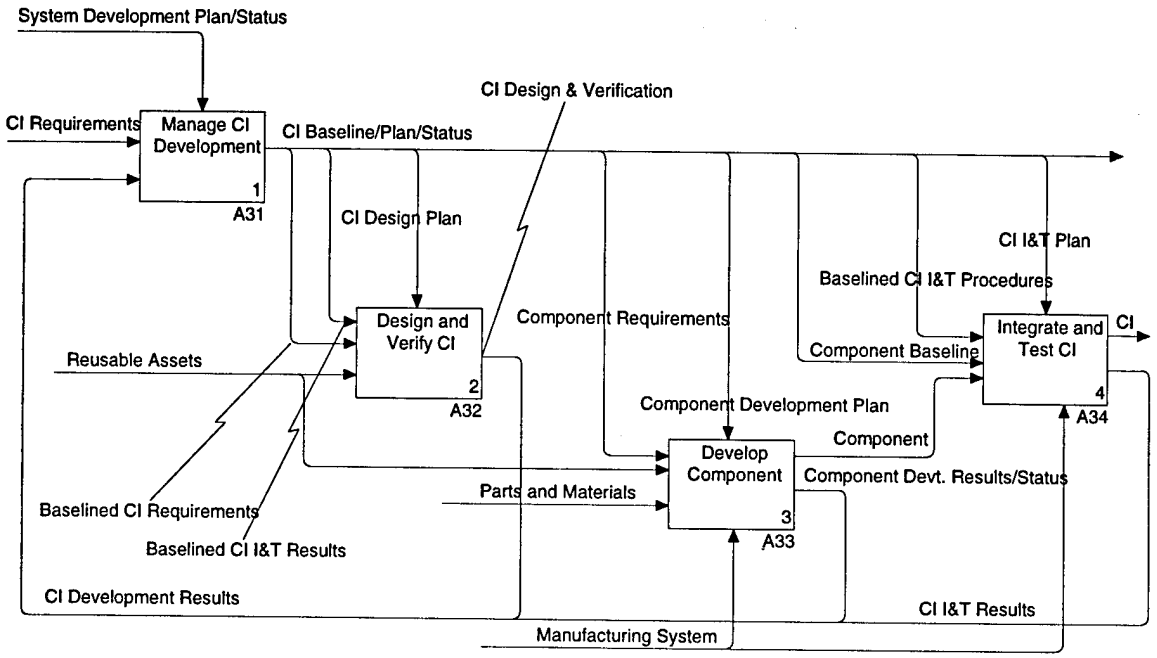


Figure 8. Develop Configuration Item (A3)

The **Develop Configuration Item** activity in the decomposition contains the following activities:

- Box 1, **Manage CI Development,** plans, controls, and coordinates the development of the CI of concern. This activity uses the *System Development Plan/Status* (produced by the **Manage System Development** activity) along with the *CI Requirements* (allocated to the CI by the **Design and Verify System** activity) as a basis for planning the CI development, including the definition of the CI development increments. The results of the design, development, and integration and test activities produce the *CI Baseline/Plan/Status*, which grows with each build of the CI.

- Box 2, **Design and Verify CI,** transforms the *Baselined CI Requirements* into a set of *Component Requirements,* taking into account the *Baselined CI I&T Results* (for the second and subsequent iterations). If there are *Reusable Assets,* this activity reviews them for use in the CI under development, as well as for reusable fragments of requirements and design specifications. In addition, it produces *CI I&T Procedures* (part of *CI Design and Verification*), which are baselined for use by the **Integrate and Test CI** activity. Note that this activity does not communicate directly with the **Develop Component** activity; instead, outputs are first baselined by the **Manage CI Development** activity.

- Box 4, **Integrate and Test CI,** assembles and tests the components according to the *Baselined CI I&T Procedures*, which include test cases and expected results, and documents the outcome in the *CI I&T Results*, which describe the status of the integration and test process and any exceptions to the observations expected by the procedures. An unsuccessful completion of this activity implies the need for reimplementation of the **Design and Verify CI** activity (and, if problems cannot be resolved there, reimplementation of the **Design and Verify System** activity as well). Successful completion of integration and test for the entire CI produces the *CI*.

The following bullet describes the component-level activity:

- Box 3, **Develop Component,** creates an integrated and successfully tested component that meets the *Component Requirements* generated in the **Design and Verify CI** activity and baselined in the **Manage CI Development** activity as part of the *CI Baseline/Plan/Status*. This activity uses the *Component Development Plan* as a control and documents any status information and risks in the *Component Devt. Results/Status*. In the case of a hardware component, the input *Parts and Materials* and the mechanism *Manufacturing System* are used to create any tangible parts. This activity also uses the adaptable components requirements and architectural and detailed designs contained in the *Reusable Assets* if they can be adapted to the component under development.

3.3.5.1 Manage CI Development (A31)

Figure 9 shows the ISSEP management activities at the configuration-item level. This activity is responsible for maintaining baselines of the configuration-item-level products and managing the configuration-item-level activities. This activity defines the configuration item context, analyzes configuration item risks, and uses this information along with the current development status to produce the configuration item development plan. The management activities are repeated every increment; thus, in every increment the context, risks, and plan are reviewed, and the plan is modified to reflect the current status and updated, as necessary, to ensure that configuration item development remains focused on critical configuration item objectives.

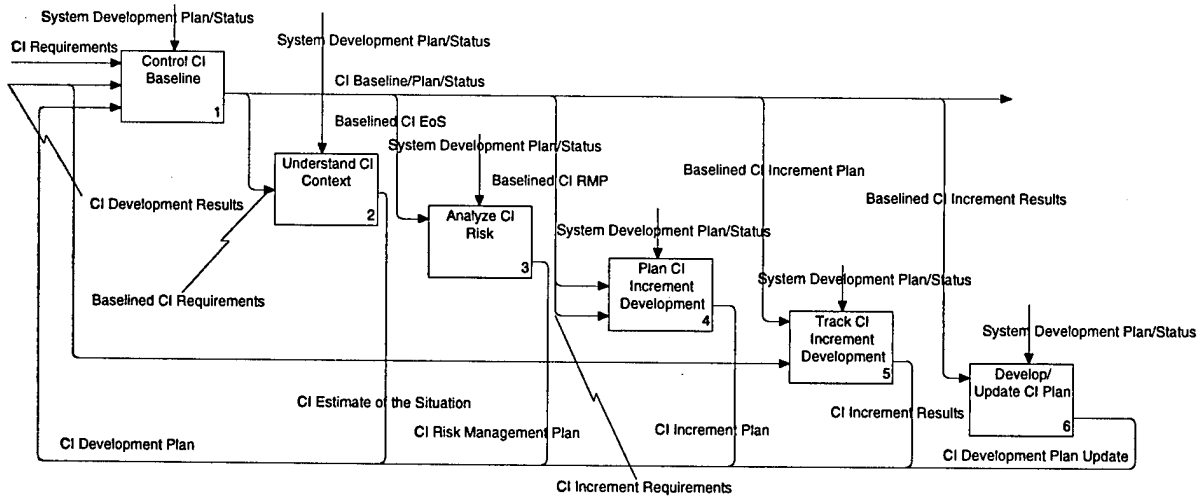


Figure 9. Manage CI Development (A31)

The following list describes the management activities:

- Box 1, **Control CI Baseline**, establishes a baseline of the CI. Every nontangible part in the CI (e.g., components, plans, and design documents), at some point, is input into this activity where it is reviewed to determine whether it qualifies for baselining and, if accepted, added to the current baseline, which is output. Even the outputs from the other management activities in this diagram are baselined in this activity. As part of the review activity, changes to the previous baseline are noted, and these changes are stored, tracked, and analyzed as part of the planning process.

- Box 2, **Understand CI Context**, identifies factors that could have an influence on the success of the CI development and defines the scope of this increment of the development. The *Baselined CI Requirements* are analyzed, along with other pertinent parts of the current *CI Baseline/Plan/Status*, to determine the increment objectives and constraints and to identify alternatives for meeting the objectives while remaining within the constraints. This information is documented in the *CI Estimate of the Situation.*

- Box 3, **Analyze CI Risk**, identifies potential increment risks, analyzes the risks to determine which are critical to the development effort and when mitigation action is recommended. The activity uses this information to develop a set of mitigation strategies for each risk and a time table for implementing the mitigation strategies. The main source used for identifying the risks is the *Baselined CI EoS,* and the output from the analysis is the *CI Risk Management Plan.*

- Box 4, **Plan CI Increment Development**, creates the detailed development plan for the next CI increment. This activity uses the *Baselined CI RMP* and the *CI Increment Requirements* to determine how to achieve the increment objectives and mitigate risk. Development goals for the increment are established and used as a basis for selecting a development strategy. Detailed size, cost, and schedule estimates are made. The development process for the increment is tailored and instantiated, and detailed work assignments are documented. This detailed planning remains within the scope of the *CI Development Plan*, adding detail, as necessary, to make the plan enactable. The output of this activity is the *CI Increment Plan.*

- Box 5, **Track CI Increment Development**, uses the *CI Development Results* to assess the progress of the development and analyze the seriousness of situations that arise. This activity controls the enactment of the *Baselined CI Increment Plan* and ensures that the development is enacted according to plan. When the development deviates too far from the plan or when the development goals documented in the plan are met, this activity terminates the development activities and initiates the **Develop/Update CI Plan** activity.

- Box 6, **Develop/Update CI Plan**, uses the *Baselined CI Increment Results* to create the *CI Development Plan.* This plan defines each of the development increments at a high level. If a *CI Development Plan* already exists because it was created in a previous increment, this activity updates the plan based on the results of the past increment development efforts, including lessons learned, newly identified risks, and status information. If this is the first increment, this activity generates the first version of this plan from the context information and risk analysis. This plan is a living document and is kept accurate and current.

3.3.5.2 Design and Verify CI (A32)

Figure 10 defines the software design and verification activities. The major input for this activity is the *Baselined CI Requirements,* output from the **Manage System Development** activity where the requirements were baselined. As with the **Design and Verify System** activity, there are three design activities that are followed by two activities for evaluation and verification. The only difference between the system-level activity and this one is that **Define Functional Architecture** and **Synthesize Physical Architecture** have been replaced with **Perform Architectural Design** and **Perform Detailed Design,** respectively. As at the system level, the results of evaluation and verification are fed back into the requirements analysis and design activities.

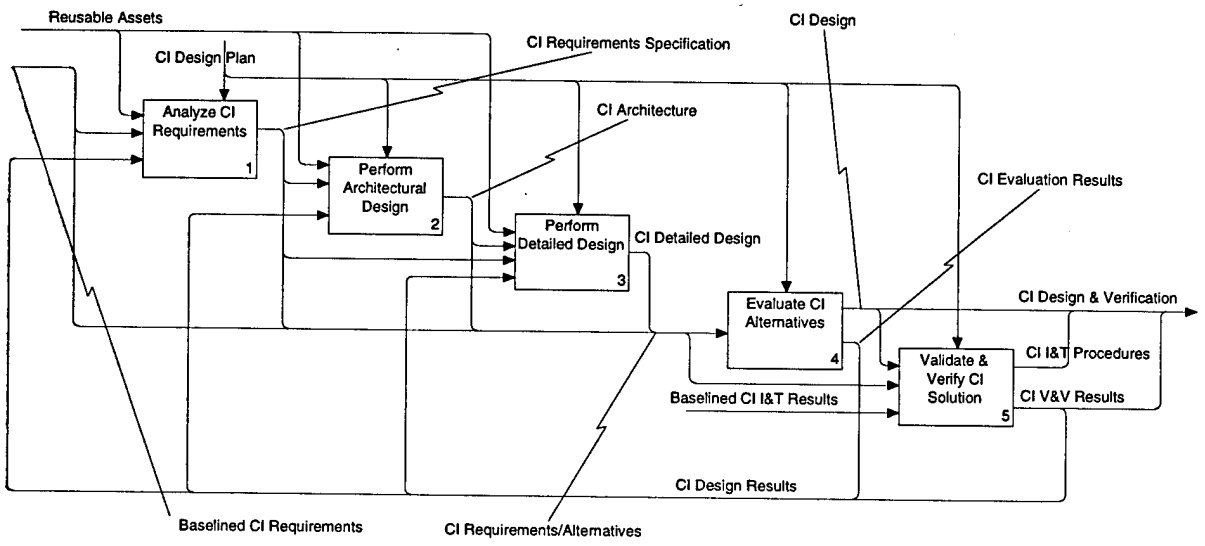


Figure 10. Design and Verify CI (A32)

The following list describes the CI-level design activities:

- Box 1, **Analyze CI Requirements,** refines the *Baselined CI Requirements* to produce a requirements specification that is usable by the design, evaluation, and verification and validation methods of choice. If *Reusable Assets* are available, this activity examines them to find existing components as well as requirements specification and design artifacts for potential reuse. The output of this activity elaborates the behavioral and performance requirements for the CI, ensuring that they are sufficiently detailed for use in the design. These requirements, if met, will ensure that the CI satisfies the *Baselined CI Requirements.* In some cases, issues arising in this activity will necessitate reimplementation of system design and/or requirements analysis activities.

- Box 2, **Perform Architectural Design,** creates a *CI Architecture* by allocating requirements to software components and describing their interrelationships (e.g., dependencies and interfaces between components). This activity uses the *Reusable Assets,* if available, to locate any components or architectural fragments to be reused. The dependencies, input/output behavior, and performance constraints (e.g., throughput, stimulus/response time) of each component are specified in this activity.

- Box 3, **Perform Detailed Design,** specifies any necessary information about the internal structure of the components identified in the **Perform Architectural Design** activity. Such information can include mandated algorithms, data structures, or code fragments (either existing or to-be-developed), details of internal logic (e.g., conditional paths of execution and the timing allocations for each), and any other constraints on the internal design. If available, the *Reusable Assets* are reviewed for any reusable specifications or specification fragments.

The following list describes the evaluation and verification activities:

- Box 4, **Evaluate CI Alternatives,** selects the requirements specification, software architecture, or detailed design that best meets the *Baselined CI Requirements* (both functional and performance requirements). It is also important that the alternative meets the constraints in the *CI Design Plan.* This activity is invoked after (and perhaps during) each of the preceding design activities; results of the evaluation may necessitate reimplementation of one or more previous activities. If at least one alternative meets all applicable requirements and constraints, it will be verified and validated in the **Validate & Verify CI Solution** activity. Evaluation results and any recommendations for improvement are communicated in the *CI Evaluation Results.*

- Box 5, **Validate & Verify CI Solution,** validates *CI Requirements* for compliance with the *Baselined CI Requirements.* This activity also verifies *CI Architecture* against *CI Requirements Specification* and verifies *CI Detailed Design* against *CI Architecture.* Both validation and verification review the input specification (including requirements traceability) for completeness and consistency. This activity generates the *CI I&T Procedures* for use by the **Integrate and Test CI** activity. After completion of that activity, **Validate & Verify CI Solution** analyzes the *Baselined CI I&T Results* to determine whether any changes to requirements analysis or design decisions are necessary to make the integrated CI ready for integration at the system level. This activity examines all the requirements to verify that the selection made in **Evaluate CI Alternatives** really does meet the requirements.

3.3.5.3 Develop Component (A33)

Figure 11 defines the activities that produce the system components. Both hardware and software components can be developed using this set of activities, but the focus of this technical report and this description is on the development of software components. The *Component Development Plan* provides the plans that control the activities. The major input into the process is the *Component Requirements* and the activities produce the component and all the design, testing, and status information that accompany it. There is a feedback loop from the **Perform Unit Testing & Analysis** activity to the other two activities. This feedback loop communicates the results of the testing and analysis so that modifications can be made to either the component or the test cases as a prelude to retesting the component. The component is released to the **Develop Configuration Item** activity only after it has adequately passed the unit testing.

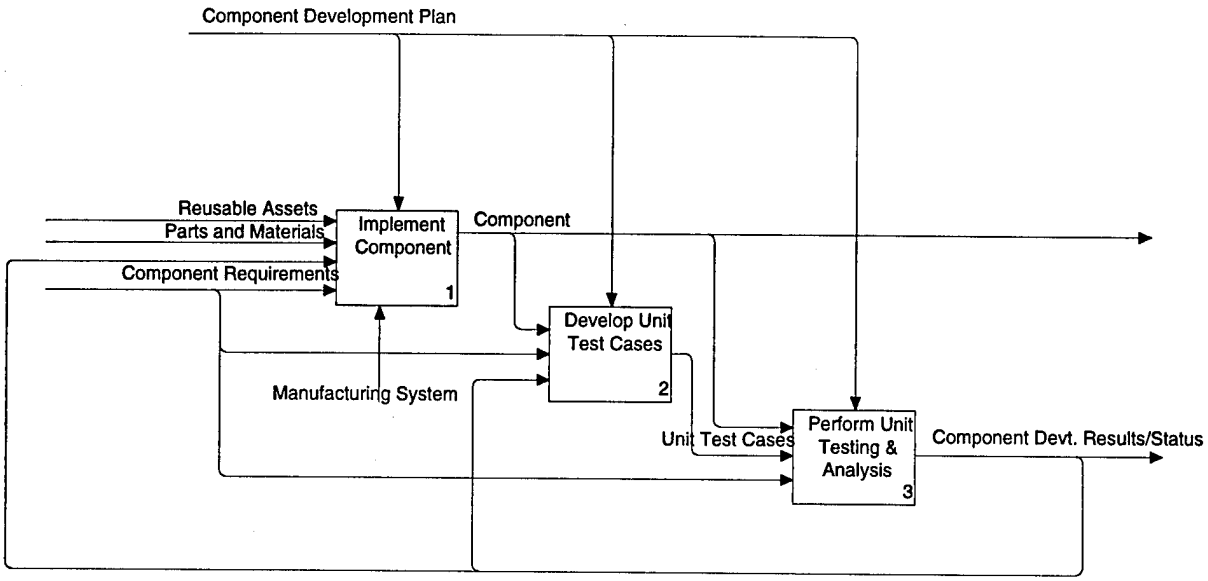


Figure 11. Develop Component (A33)

The following list describes the design activities:

- Box 1, **Implement Component**, produces the component, given the *Component Requirements*. The *Component Development Plan* describes how the component will evolve (if at all) over multiple builds of the CI. For software components, this activity equates to coding. For hardware components, this activity may involve using a manufacturing system to assemble physical *Parts and Materials*. If *Reusable Assets* are available, they are incorporated in the implementation as specified by the *Component Requirements*. The *Component* that is produced includes implementation decisions, status, lessons learned, and newly identified risks that may impact future development or evolution of the component or the development and/or integration of other parts of the CI or another system part.

- Box 2, **Develop Unit Test Cases**, produces the test cases and specifies the order in which they will be run. The result is output as *Unit Test Cases*, which are used for unit testing. This activity follows **Implement Component** because unit testing is usually "white box." As used here, white box means that the *Unit Test Cases* are based on both *Component Requirements* and the structure of the *Component* itself. "Black box testing," a subset of white box testing, is based only on requirements. White box testing includes all the black box cases plus additional cases based on design. The *Component Development Plan* specifies whether white box or black box testing should be performed.

- Box 3, **Perform Unit Testing & Analysis**, implements the *Unit Test Cases* on the *Component* and analyzes the test results to ensure that implementation of the component is complete and consistent with respect to *Component Requirements*. If not, it will be necessary to repeat one or both of the previous activities. The *Component Development Results/Status* are produced. These results include the implementation decisions and rationale; the test cases; the results of the testing and the associated analysis; any newly identified risks; and the source code, if this is a

software component. The *Component* is a hardware unit if this is a hardware component. The *Component* is an executable version of the software unit on an appropriate electronic media (e.g., tape or diskette) if this is a software component.

3.3.5.4 Integrate and Test CI (A34)

Figure 12 defines the activities in which the software components are assembled into the CI and tested to ensure that they meet requirements. The major inputs for this activity are the *Baselined CI I&T Procedures,* produced by the **Validate & Verify CI Solution** activity and baselined by the **Manage CI Development** activity; the *Component Baseline*; and the collection of *Components* to be integrated and tested. Results of this activity are a *CI Aggregate* (or the entire *CI)* and the *CI I&T Results,* which are baselined by **Manage CI Development** and fed back into the **Validate & Verify CI Solution** activity, if necessary.

In this activity, the I&T procedures are detailed based on design information; the CI is assembled, integrated, and tested; and the integration testing is analyzed.

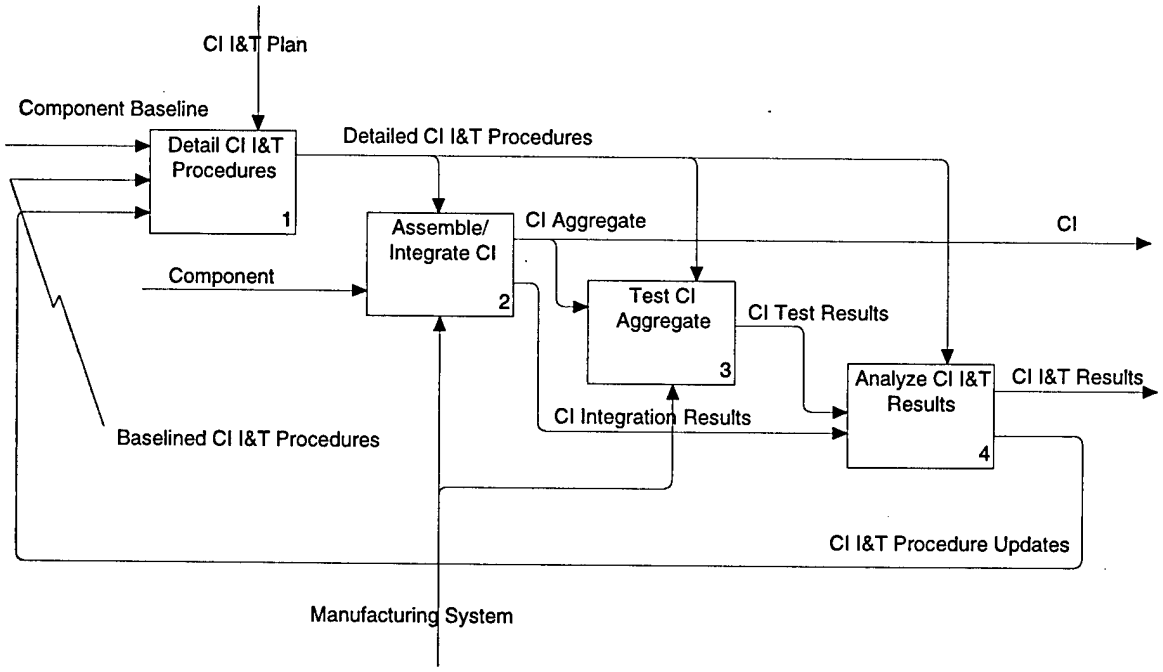


Figure 12. Integrate and Test CI (A34)

The following list describes the activities for integrating and testing software components within one CI:

- Box 1, **Detail CI I&T Procedures,** transforms *Baselined CI I&T Procedures* into *Detailed CI I&T Procedures* based on design information in the *Component Baseline*. If there are multiple iterations of testing one CI, this activity will also use the *CI I&T Procedure Updates* resulting from analysis of testing results. The *Detailed CI I&T Procedures* should be sufficiently detailed to allow integration and test engineers to carry out the **Assemble/Integrate CI** and **Test CI Aggregate** activities.

- Box 2, **Assemble/Integrate CI,** assembles individual *Components* into a *CI Aggregate* (which may be the entire *CI).* The components are assembled according to the detailed procedures defined in the **Detail CI I&T Procedures** activity. Results of this activity are the *CI Aggregate* to be tested and the *CI Integration Results* for use in analyzing test results.

- Box 3, **Test CI Aggregate,** carries out the tests prescribed by the *Detailed CI I&T Procedures* on the *CI Aggregate.* If necessary, this activity also involves inspecting commercial off-the-shelf (COTS) or other reusable components (i.e., anything not produced by **Design and Verify CI)** to ensure that they will work properly with other components. Note that in this activity, you inspect the actual reusable software or hardware; in **Validate & Verify CI Solution,** the designer (presumably) reviewed their documentation. This activity produces *CI Test Results* for analysis in the next activity.

- Box 4, **Analyze CI I&T Results,** reviews the *CI Integration Results* and *CI Test Results* to see if any problems were uncovered. If so, this activity will determine whether the problem lies in testing and integration. If so, it will define *CI I&T Procedure Updates* and cause the entire **Integrate and Test CI** to be repeated. Otherwise, it will capture the results of integration and testing in *CI I&T Results,* which will be reviewed by management, baselined, and provided to the **Validate & Verify CI Solution** activity.

3.3.6 INTEGRATE AND TEST SYSTEM (A4)

Figure 13 defines the system integration and test activity in the ISSEP model. This activity assembles and tests the hardware and software CIs according to the *Baselined System I&T Procedures*, which include test cases and expected results, and documents the outcome in the *System I&T Results*, which describe the status of the integration and test process and any exceptions to the observations expected by the procedures. When this activity has been successfully completed, the *Operational System* is ready for delivery to the customer. The *Operational System* is the tangible part of the system, but may contain nontangible items such as software and user and design documentation.

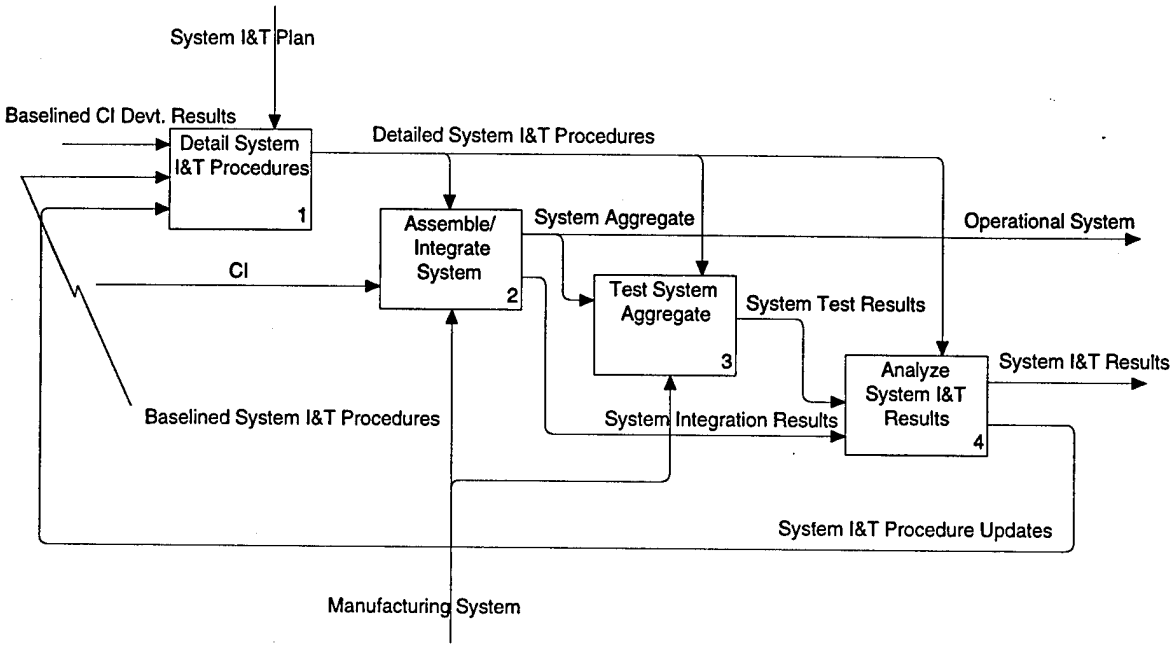


Figure 13. Integrate and Test System (A4)

The following list describes the integration and test activities:

- Box 1, **Detail System I&T Procedures,** uses implementation details from the *CI Designs* and the *CI I&T Results* and uses parts of the *Baselined CI Development Results* in the *System Baseline/Plan/Status* to elaborate on the *Baselined System I&T Procedures*. This activity takes into account any *System I&T Procedure Updates* that may have been fed back by the **Analyze System I&T Results** activity and produces a set of *Detailed System I&T Procedures* explaining exactly how the CIs are to be integrated and tested and how the results are to be analyzed.

- Box 2, **Assemble/Integrate System,** progressively assembles and integrates the CIs according to the *Detailed System I&T Procedures* until the *Operational System* is complete, then provides the *System Integration Results* for analysis.

- Box 3, **Test System Aggregate,** if needed, inspects incoming CI-level COTS parts. After each step of integration, as defined by the *Detailed System I&T Procedures*, this activity performs the specified tests on the current *System Aggregate*, up to and including the *Operational System* itself. Finally, this activity uses independent personnel to perform any required system qualification testing and provides the *System Test Results* for analysis.

- Box 4, **Analyze System I&T Results**, uses the *System Integration Results* and the *System Test Results* to determine whether the *Detailed System I&T Procedures* need to be updated. This activity lists any failures and their apparent causes and generates the *System I&T Results* to be baselined and forwarded to the validation and verification activities of **Design and Verify System.**

3.4 SUMMARY

This section described the ISSEP model. Descriptions of each activity and how the activity consumes the inputs to produce the outputs have also been included. Two levels of system decomposition were used to illustrate the application of the model: from System to CI and from CI to Component. This decomposition was matched by levels of the model, the framework for which was provided in the activities **Develop Operational System** and **Develop Configuration Item**. The point was made in Section 3.1 that, in general, the system of interest may be decomposed into an arbitrary number of levels, elements of which may have such names as "unit," "subsystem," and "segment," and may consist of software or hardware or both. The ISSEP model may then be instantiated to correspond to those levels.

The major activities represented by the ISSEP model at any given level, other than the bottom level, are shown by Figure 14 in a building block diagram called "Develop Level (n)," which is a generalization of the **Develop Operational System** and **Develop Configuration Item** diagrams. The **Develop Component** diagram represents the bottom or "leaf" level of development even in the general case.

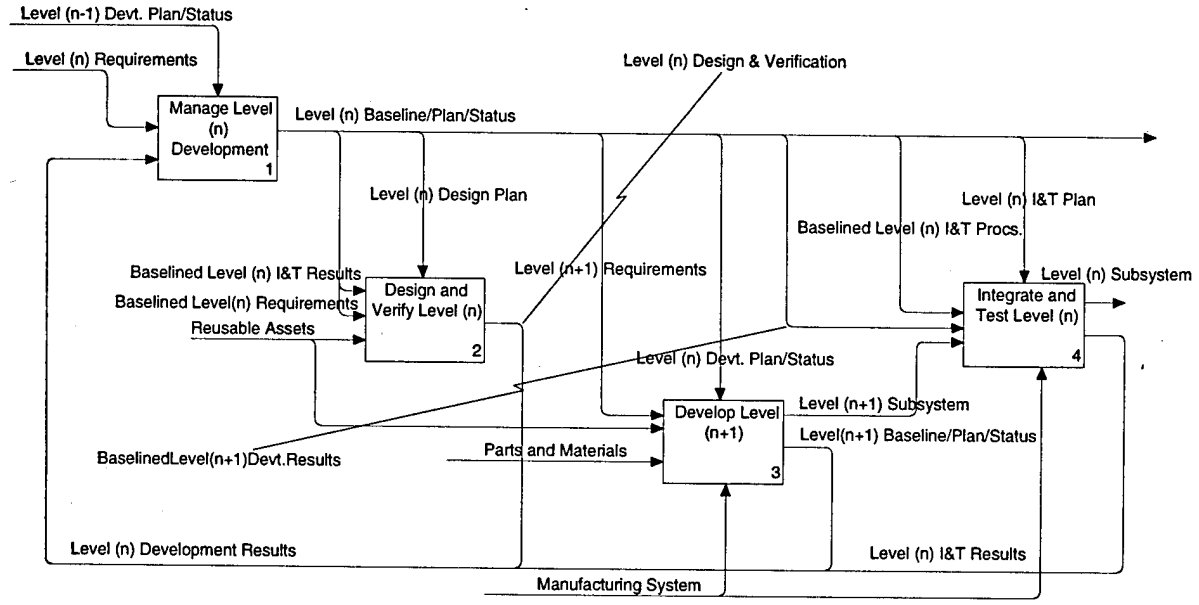


Figure 14. Building Block: Develop Level (n)

4. ISSEP CONCEPTS AND RATIONALE

A greater appreciation of the ISSEP model can be achieved by understanding the underlying model concepts in addition to an examination of the model itself. This section describes the following major ISSEP model concepts and includes the rationale for why and how these concepts are critical:

- Systems and software interfaces
- Information flow
- The development plan
- Risk management

4.1 SYSTEMS AND SOFTWARE INTERFACES

The ISSEP model distinguishes between the systems and software process interfaces and the systems and software product interfaces. The process interfaces define the information flows between software- and system-centered activities. The product interfaces define the points within the developing system where the system and software parts interact. The following list describes the ISSEP model's process and product interfaces:

- ***Process Interface.*** The information flows in the ISSEP model define the activity order. The flows are designed such that the systems design is strongly tied to the software design. The system-level design information flows to the software level where it is used to produce the software design. In return, risks associated with the software design, requests for information, and design decisions flow from the software level to the system-level design. The system design cannot be finalized until it receives confirmation that the software level can support the system-level design decisions. This is an iterative process, where information is exchanged until consensus on the design is reached.

  The system may be decomposed into several levels of design. Software-intensive systems may include several levels of software below the systems level. Regardless of the number of levels, the lowest level software design and risks become incorporated into the design at all levels in the decomposition by being incorporated by the next higher and higher levels, one level at a time. Indeed, design and risks from one subsystem may even influence the design of other subsystems because they force a change in the allocated requirements and/or implementation strategy.

  One important aspect of the ISSEP model is to ensure that integration risks are evaluated and proactively dealt with during design. Integration of software with software, software into system

parts, and system parts with other system parts is examined and incorporated at all levels of the design. Then, at every level in the decomposition, the results of the integration are evaluated to identify any risks that may create future integration problems (either in the remainder of this development effort or in the extended life of the system).

- ***Product Interfaces.*** The system/software product interfaces are established iteratively during the design activities, but are finalized during the **Synthesize Physical Architecture** activity. Although this is a system-level activity, the product interface designs are significantly influenced by the software designs. Indeed, because of the recursive nature of the model, the software interface designs are often agreed upon before the system interface designs have finished negotiation. This is not the traditional ordering where the interfaces are defined by the system-level activities and software design activities must implement them. Unlike the traditional ordering, the ISSEP model ordering encourages commitment and buy-in from affected development groups, which has psychological benefits. In addition, the ISSEP model's ordering has the advantage of ensuring that the software design has had ample opportunity to influence the total system design, which decreases the number of system/software integration problems.

4.2 INFORMATION FLOW

Information flow is the key to how the ISSEP model activities are implemented. This section describes the following information flow concepts:

- Partial ordering of activities
- Formal/informal and major/minor information flows
- Pull versus push philosophy
- Tightly coupled communications with reply
- Management control of information flows

4.2.1 PARTIAL ORDERING OF ACTIVITIES

The layout of the activities in the IDEF0 diagrams in the ISSEP model does not represent time sequencing or dependencies; rather, the layout defines the information flows between activities. The objective of the layout design is to make information flows as clear and easy to follow as possible. However, information flows do define a partial ordering of the ISSEP activities. The rules for the partial ordering determine when ISSEP activities can begin and end.

An ISSEP activity begins whenever the inputs are available and ends when the outputs are completed. The complete input is not required before the activity begins as long as the part of the input necessary for commencing work on the activity is available. For example, the *Risk Management Plan* is an input to the **Plan System Increment Development** activity. The entire Risk Management Plan (RMP) need not be complete before the planning activity can begin. In fact, frequently the development plans do not include the risk mitigation activities initially. Later, the risk mitigation activities are added to the plan, and their impact to the plan is analyzed and used to help in mitigation selection. Thus, information about the plan influences the creation of the RMP, and RMP influences the plan contents. However, the RMP must be

completed before the *Increment Plan* can be completed because the risk mitigation strategies documented in the RMP must be adequately represented in the *Increment Plan* for effective risk management to occur.

The information flows in the ISSEP model are partially ordered because the input information flows are not required to be complete before initiation of an activity. The inputs need to be complete only before the outputs are completed.

There is, however, one other factor that determines when the ISSEP activities begin and end: the development plans that control the activity. These plans contain information that may delay the start or end of an activity. For example, the **Analyze System Risk** activity is controlled by *the Organizational Plans/Status*. This plan directs the **Manage System Development** activities and provides coordination information, such as availability of risk analysis information from other processes (e.g., CI risks) and also controls the **Analyze System Risk** activity so that other parallel, concurrent processes can have access to this information. This coordination is necessary to ensure that information needed in different parts of the system development is available for timely decision making. Section 4.2.5 gives more detail on how the management activities control the information flow.

4.2.2 Formal/Informal and Major/Minor Information Flows

Processes have formal and informal information flows. A formal information flow is one that is required by the process. An informal information flow is one that is not required by the process, but that may be used for convenience. Informal information flows may never substitute for the formal ones defined in the process. An example of a formal information flow is the requirement for each set of development activities to provide status information to the management activities. An example of an informal information flow is the transfer of status information from one set of development activities directly to another set of development activities.

Integrated product development teams are a mechanism sometimes used for implementing formal and informal information flows. The ISSEP model defines only the formal interfaces. The ISSEP model does not preclude the use of informal interfaces, where appropriate, but those interfaces are not included in the ISSEP model descriptions. Eliminating the informal interfaces from the model not only reduces the arrows on the diagrams, but focuses the reader on the primary communication among activities.

Information flows can also be either major or minor. A major information flow contains information critical to the implementation of the process. All of the information flows in the ISSEP model descriptions are major. Most information in the process is transferred along a few major flows.

Minor information flows meet at least one of the following criteria:

- Information not critical to the implementation of the process. This information is excluded from the ISSEP model to improve the clarity of the diagrams.
- A flow that is not consistently produced or consumed. These flows may be added to the ISSEP model as part of a specific tailoring of the process, as required for a particular situation.
- Information assumed present. For example, engineers' access to a technology base is assumed and, therefore, not explicitly included in the ISSEP model.

An example of a minor information flow is status information from the **Analyze System Risk** activity. This information could contain such items as the total effort spent performing the analysis and how many people were involved. This type of information is assumed present. When minor information flows are needed, they are specified during process tailoring (see Sections 5.1 through 5.3).

4.2.3 PULL VERSUS PUSH PHILOSOPHY

There are two opposing views of how information flows into activities: information can be pushed or pulled into the activity. In the pull mode, activities "pull" needed information from a source. In the push mode, the activities receive everything possible from the source, and must distinguish mandatory information from optional information based on activity needs. The ISSEP model can be characterized as a "pull" model because it only identifies the mandatory information that flows into an activity; any additional information, if necessary, must be pulled from the source. The ISSEP model designates the mandatory information so that there is no confusion in the receiving activity as to what inputs must be included in the creation of the mandatory outputs.

For example, in the **Develop Operational System** diagram (A0), only the *CI Requirements* are mandatory inputs to the **Develop Configuration Item** activity. If, however, the **Develop Configuration Item** activity needs information that is part of the *System Baseline* but not part of the *CI Requirements*, it can pull that information from the *System Baseline* flow. Specifically, if the **Develop Configuration Item** activity needs to examine the proposed test cases contained in the *Baselined System I&T Procedures*, the activity pulls that information through the *CI Requirements* information flow. (Note that *Baselined System I&T Procedures* is a major information flow from the perspective of the **Integrate and Test System** activity.) However, most of the time the test cases will not be required and are not required inputs to the **Develop Configuration Item** activity.

4.2.4 TIGHTLY COUPLED COMMUNICATIONS WITH REPLY

Some information flows from an activity contain requests for action. For example, risks identified during the implementation of an activity need to be resolved, and the resolution needs to be communicated back to the initiating activity. The risk information may also be sent with or without the identified resolution implementation information to other affected activities that are required to respond and indicate what action was taken. This type of information flow is necessary to ensure that information reaches the appropriate destination and the necessary actions have been taken.

The ISSEP model contains this type of feedback loop and has labeled this type of information flow "tightly coupled communications with reply." The following example demonstrates this type of loop in the **Develop Operational System** diagram. The *CI Baseline/Plan* that is output from the **Develop Configuration Item** activity may contain requests for action (e.g., requests for more parts, more personnel, a modification to the current schedule). These requests flow into the **Manage System Development** activity where resolutions are either made or the requests are passed via the *System Baseline* or the *System Development Plan/Status* to other activities that may be able to honor the requests. Regardless of which activity(ies) ultimately resolves the issues, the resolution(s) is documented by that activity, action is taken, and the action and results are passed back to the **Develop Configuration Item** activity from the resolving activity through the **Manage System Development** activity as part of either the updated *System Baseline* or *System Development Plan/Status*. It is the responsibility of the

Manage System Development activity to track the status of the reply and to ensure that the reply is sent to the **Develop Configuration Item** activity.

4.2.5 MANAGEMENT CONTROL OF INFORMATION FLOWS

Information in the ISSEP model flows to and from the set of management activities. The management activities are the control point in the process because they:

- Establish baselines for release to other activities (i.e., create and maintain the baseline flows)
- Control the routing of information to other activities
- Use the information to update and maintain the development plan

The concept of management activities being the control for the development activities is not unique; however, the ISSEP model's definition of the management activities and management philosophy is worthy of explanation.

Management can mean different things: it can be a role, a function, or a set of activities. In the ISSEP model, management refers to the set of management activities. Managers (the role) perform some of the ISSEP management activities, but not all. For example, a risk analyst, not the project manager, may be assigned to the **Analyze System Risk** activity. Many of the functions typically associated with management are included in the ISSEP management activities (e.g., planning and tracking); however, other management functions, such as generating performance reviews, are not part of the ISSEP management activities. Other functions not typically associated with the management function, such as controlling a baseline, are included in the ISSEP management activities.

In the ISSEP model, the management activities are decentralized. Each part of the system decomposition (except the component level, which is not further decomposed) has its own management activities. This helps ensure communication flow by removing the bottleneck situation that can result from a central management control point (in much the same way that decentralizing computing processors eliminates the bottleneck of trying to access a common processor). Decentralization also improves visibility (i.e., managers can be focused on the small set of activities within that system part and not be distracted by the overall complexity of the development effort), which results in better planning and decision making. In this way, plans and decisions can be very specific and incorporate very detailed information that might be lost in the masses of information flowing through a centralized set of activities. The management activities route information to other system parts to ensure that the management activities' narrow visibility does not lead to local decision making that negatively impacts the system as a whole.

4.3 THE DEVELOPMENT PLAN

The management activities two main functions are controlling information flow and creating and maintaining the development plan. The ISSEP model defines the development plan as a living document that is continuously updated and expanded to provide accurate and current plans. The development plan is composed of a long-term plan and an increment plan.

4.3.1 LONG-TERM PLAN

The scope of the long-term plan is the entire system (or system part) being developed. (There is a development plan for each level in the system hierarchy). This plan defines the context for the development (i.e., scope, objectives, and stakeholders), documents constraints levied from above (or from an external source), defines standards and organizational processes to be used, defines the work products to be produced, defines a work breakdown structure, defines the system parts (e.g., CIs) that will make up this system part, and defines the increments (e.g., subsets). The long-term plan is a high-level plan. Detailed planning is contained in the increment plan. In general, the long-term plan provides the road map of what will be done and the plan for making sure that all the system parts come together in the end.

The long-term plan for a system part is influenced by constraints from system parts above and feedback from system parts below in the system hierarchy. For example, a CI's long-term plan is constrained by its requirements and scope that were determined by its parent system part (i.e., the system part above it in the hierarchy). Within those constraints, the CI creates a long-term plan that defines how it will be decomposed (in addition to the other parts of the long-term plan listed above). If that decomposition defines three components, each of the components provides feedback to the CI describing their long-term plans and indicating risks associated with the CI's long-term plan. The CI modifies its long-term plan, as appropriate, and sends feedback to its parent system part containing its revised long-term plan (if it was revised) and how the constraints originally passed down to it may be affected if the long-term plan is implemented. Thus, the long-term plans from each system part in the hierarchy influence each other and converge on a realistic plan for the total development effort.

4.3.2 INCREMENT PLAN

The long-term plan constrains the increment plan, which defines how a particular system part will be developed (e.g., the sequence of activities that need to be performed and when they should be performed to meet the objectives of the long-term plan). Not all the increment plans are defined at the same time. The ISSEP model encourages plans to contain the detail for the next increment (or several increments), but not for all future increments. Too much detailed planning up front usually leads to rework of plans or inflexibility. As the time for implementing future increments approaches, the increments are expanded to include more and more detail until, when the implementation begins, a very detailed plan that is specified to an enactable level of detail is available.

Just as each system part has a long-term plan, each system part has an increment plan. The content of the increment plans of one system part is affected by the increment plans from other system parts above it and below it in the system hierarchy in the same way that the long-term plans of the system parts affected each other. However, less iteration is needed when elaborating the increment plans because the major negotiations have already taken place when the long-term plans were defined because the long-term plans establish the context for the increment plan development.

Increments define the partitioning of work into manageable pieces. However, the size of a manageable piece differs depending on the complexity. Because the definition of an increment is so dependent on the specific characteristics of the system part being developed, the ISSEP model defines the increments for each system part dynamically during the implementation of the management activities. That is, during the initial implementation of the **Manage System Development** activities, the increment definition for the entire system is created at a high level. (The increment definitions are contained in the long-term

plan.) This increment definition is passed down to the **Develop Configuration Item** activity and into the **Manage CI Development** activity, providing the basis for defining the increments for the CI.

As an example, the long-term plan (from the **Manage System Development** activity) defines the system increments as shown in Table 2.

Table 2. System Increments

Number	Description
Increment 1	Develop system development plan
Increment 2	Design system
Increment 3	Develop high-risk CIs and any items with tightly coupled dependencies
Increment 4	Integrate and test the developed CIs
Increment 5	Develop remainder of the CIs and make any needed modifications
Increment 6	Integrate and test entire system

After the first and second increments have been completed and the system design is known, assume the system is decomposed into the following four CIs: CI1, CI2, CI3, and CI4. Table 3 shows the updated system Increments 3, 4, 5, and 6.

Table 3. Updated System Increments

Number	Description
Increment 3	Develop CI1 and CI3 (assume CI1 is a high-risk component and CI3 is tightly coupled to CI1)
Increment 4	Integrate and test CI1 and CI3
Increment 5	Develop CI2 and C14
Increment 6	Integrate CI1 and CI3 with CI2 and CI4

Each CI would create an increment plan that details how the increment will be developed. Each increment definition has associated constraints (e.g., cost and schedule) that are documented in the long-term plan. Using the above example, CI1 and CI3 receive the *CI Requirements* and the increment constraints from the management activities at the beginning of the third increment. This information constrains the increment plans created by CI1 and CI3 for their development, which takes place in the system's third increment. CI1 and CI3 have their own increment definitions (numbered 1 to n), which are based on the definition of the system's Increment 3 in Table 3. For example, during Increment 1, the management process for CI1 defined the long-term plan, including objectives, constraints, cost, schedule, increment definitions, and development process for CI1's development. Table 4 lists the increment definitions in CI1's long-term plan.

Table 4. CI1 Increments

Number	Description
Increment 2	Complete CI requirements
Increment 3	Complete CI design
Increment 4	Develop all CI components
Increment 5	Integrate and test components

Note: CI1's increment plan did not include Increment 1 because Increment 1 was the increment that created the long-term plan.

The management process for CI3 may develop a different increment plan. It is not required that the increment plans be similar as long as they meet the constraints of the system increment plan.

There are two requirements for increment definitions in the ISSEP model: an increment must 1) move the development toward completion, and 2) contain at least one implementation of the management activities. In the example shown in Tables 2 through 4, after the technical activities of each increment are complete, the following management activities are performed:

- ***Track Increment Development***. This activity is started at the beginning of the increment, but does not complete until the increment ends.
- ***Develop/Update Plan***. The system plan is modified, if necessary, based on the results of the development done during this increment.
- ***Control Baseline***. The work products produced during the increment are baselined, and the updated system plan is also baselined.
- ***Understand Context***. The context for the next increment defined in the updated system plan is analyzed and refined.
- ***Analyze Risk***. The risks associated with the next increment are identified, and risk mitigation activities are selected.
- ***Plan Increment Development***. The detailed plan for the next increment is produced.

The **Track Increment Development** activity initiates and the next increment development begins.

After each increment is implemented, the increment definition is further refined to reflect the knowledge gained during its implementation. This refinement helps the system development effort remain in control and ensures that plans remain realistic and reflect the current status.

The increment plan, the detailed plan that describes how each of the increments are to be enacted, is based on the increment definitions. In Table 4, the increment plan for Increment 2, Complete CI requirements, would define what activities needed to be performed to complete the requirements definition (e.g., **Analyze CI Requirements, Evaluate CI Alternatives,** and **Validate & Verify CI Solution**). Each of these activities would be further specified through process tailoring (see Sections 5.1 through 5.3). That is, the tailoring would specify a requirements analysis method, the tools to be used (if any), the metrics to be collected, the standards to be followed, and possibly a template for the requirements traceability matrix. The increment plan would also contain the information needed for instantiation of the specified activities including staff assignments, specific milestones (or inchstones), and logistical information such as needed facilities or technical support.

The long-term plan and increment plan work together to create a complete and detailed plan, but eliminate the requirement for the plan developers to do the detailed planning before the long-term, high-level plan is stable.

4.4 RISK MANAGEMENT

Risk is one of the important factors that influence the enactment of the ISSEP model. Risk factors influence decision making and are explicitly addressed in the project plans. Risk identification is explicitly performed as part of the **Analyze System Risk** and **Analyze CI Risk** activities. In these activities, all possible areas of risk (e.g., organizational, process, development) are examined, and risks are identified. However, risk identification and communication is an implicit (ongoing) part of all the ISSEP activities. For example, when design alternatives are evaluated (during the **Evaluate System Alternatives** activity), risks associated with the alternatives are identified and the alternative's risk profile (i.e., identified risks, risk factor estimates, and assessment of risk mitigation actions) is considered as part of the evaluation. The risk profile associated with the recommended alternative becomes part of the design documentation.

Less obvious are other, nondesign-oriented risks that may also be identified when design alternatives are evaluated, such as staffing risks. For example, the risk that implementation of the alternative requires staff with a particular skill, which is in short supply and may not be available to meet the schedule, impacts the design if it makes an implementation alternative high risk.

This implicit risk identification is part of all ISSEP activities, and the rigor of the identification is determined by the development plan that controls the activity. Thus, criteria such as the number and complexity of interfaces and the criticality of the system part are used to determine the risk identification rigor that is necessary. Determining the risk identification rigor for all system parts developed below a particular part of the system is part of the initial risk analysis and planning. Not all system parts being developed need to have the same amount of risk identification rigor (not all parts of the development are at equal risk). During the risk analysis and planning activities for every increment, the risk identification information that is received from the system parts below is evaluated, and a decision is made to have the rigor associated with that portion of the development increased, decreased, or remain unchanged. The control of risk identification rigor by the higher level system parts provides a mechanism to ensure that sufficient information is available for effective risk management while not unnecessarily burdening the development effort if risks are under control.

This page intentionally left blank.

5. APPLYING THE ISSEP MODEL

The goal in applying any engineering process is to provide sufficient guidance so the engineering can be completed in a controlled manner while minimizing the amount of unnecessary work. That is, a detailed process provides additional insights just like a detailed plan gives both the planner and the follower more information and a better understanding of what it will take to enact the plan. However, it takes time to elaborate a process in detail. Consequently, the goal is to provide sufficient detail in the process so that the process activities can be efficiently and effectively performed, but balance this goal against the effort necessary to specify the process so that the most benefit (from enacting the process) can be derived from the minimum effort (in creating a detailed process).

In reality, a perfect balance may not be achieved at first or even after several tries. Part of what makes this balancing act so difficult is that it depends on the project's unique characteristics as well as on the process. For example, more experienced engineers need less detail in the plans than do less experienced engineers, and complex interfaces require more detail in the interface design and more iteration to complete than do simple interfaces. The act of further specifying a process or process model so that it reflects the project characteristics is called process tailoring (Software Productivity Consortium 1995b).

5.1 PROCESS TAILORING

Process tailoring is creating a specific process from a general process. Process tailoring includes:

- ***Process Architecting.*** Process architecting is defining a set of processes and the interfaces between the processes. In the ISSEP model, process architecting involves defining the specific processes for the systems, subsystems, CIs, and components that make up the system being developed.

- ***Process Specification.*** Process specification is defining an activity by identifying the specific information that must be contained in the inputs and outputs, decomposing the activity into subactivities or tasks, and instantiating the activity to make an enactable process. An activity is enactable when there is sufficient detail to allow the process to be carried out by the resources assigned to complete it.

Process tailoring creates the detailed process that becomes the essence of the project plan. The tailored process defines each of the activities to the enactable level, describes precedence order of the activities and their dependencies, and describes the contents of the information flows (e.g., inputs and outputs). A large part of generating a detailed project plan is process tailoring. High-level plans such as the long-term plans described in Section 4.3 include high-level processes. Detailed plans such as the increment

plans, also described in Section 4.3, include tailored processes. The more detailed the plan, the more precisely the process is architectured and specified. Thus, process tailoring is a large part of planning.

Because every application of the ISSEP model has unique characteristics, it is not possible to define a tailored version of the ISSEP model that is generally applicable. Certainly, one approach for creating a more tailored ISSEP model is to use the IDEF0 diagrams and decompose each of the ISSEP model activities into their constituent activities (e.g., decompose **Analyze System Risk** into activities like Identify Risks, Evaluate Risks, Develop Mitigation Strategies, and Plan Risk Mitigation). It is beyond the scope of this report to describe all the concepts involved in process tailoring and process engineering (Software Productivity Consortium 1994a). It is, however, possible to describe how to apply the ISSEP model by describing what factors most influence its tailoring and how to approach introducing the model to new and ongoing development efforts.

5.2 FACTORS THAT INFLUENCE PROCESS TAILORING

The major factors that influence the tailoring of the ISSEP model are:

- Project size and complexity
- System architecture and organizational structure
- Project and process familiarity
- Project domain (e.g., real-time, information systems, C^3I)
- Project risk

5.2.1 PROJECT SIZE AND COMPLEXITY

The project's size and complexity are important factors in determining the information flow contents. The project's size can vary from a small team of individuals working together to perform the activities, to a large number of teams or individuals working independently on parallel efforts. If, for example, a small team of engineers is designing a system part, and the same team members define the functional and physical architecture and perform the evaluation of the design alternatives, documentating the rationale for each design alternative may not be necessary. Documentation of the rationale may not be necessary because team's involvement in the creation of the alternatives ensures their familiarity with designs and associated rationale. It is still necessary, however, to document the design and rationale for the selected alternative as part of the final system design.

Project complexity also impacts the amount of detail necessary in the information flows. Returning to the above example, the small team may be required to document the design rationale for every design alternative even if they perform the evaluation and if the design includes sufficient complexity. The team may forget or misinterpret their design rationale during the evaluation if it contained subtle, intricate information that might necessitate reconstruction of the missing information.

5.2.2 System Architecture and Organizational Structure

The architecture of the system being developed and the structure of the organization responsible for the development are influential in determining the ISSEP process architecture. Certainly, the system architecture determines the decomposition of the system, which is reflected in the process architecture. That is, a system may be decomposed into five CIs based on the results of the **Design and Verification** activity. The process would be architected with five **Develop Configuration Item** activities to reflect the five CIs and the interfaces between them and the system.

The organizational structure further constrains the process architecture by imposing organizational characteristics on the process. For example, two different organizational structures might be developing one of the five CIs mentioned above. In this case, it may be necessary to have that CI divided in to two different activities. That is, there would be six **Development Configuration Item** activities in the process architecture instead of five; and the two that were derived by splitting the one CI into two sets of processes may be defined slightly differently from the other four **Develop Configuration Item** activities. These two processes would be tightly coupled and may contain additional coordination and control mechanisms and plans to transfer the results of the development of one part of the CI to the other CI for final integration and test.

5.2.3 Project and Process Familiarity

Familiarity with the project and the process impacts how the process is specified. For example, if the designers are familiar with the design for this or similar projects, there is less need to document the detail associated with the design. If, on the other hand, the development is unprecedented, then the design information requires more detail to ensure that all critical design aspects are being considered and communicated. Process familiarity has a similar impact on process tailoring. The more times engineers follow a process, the more knowledgeable they become about enacting the process and the easier it is for them to efficiently perform the activities and produce the desired outputs.

5.2.4 Project Domain

The domain of the project also impacts the process specification. For example, a real-time system will require more rigorous performance and timing information than an information system. The information system may require design of data structures not required for the real-time system. These different domains impact the development methodologies selected.

5.2.5 Project Risk

Risk is one of the main drivers of process tailoring. Indeed, the factors described in Sections 5.2.1 through 5.2.4 could be generally grouped under project risk. For example, the risk that complex interfaces may be misinterpreted necessitates the addition of detailed interface descriptions to the design information. The risk that a single CI developed in parallel by different organizational structures may not be easily integratable necessitates additional process mechanisms to coordinate, integrate, and test the completed item. However, the factors described above that influence process tailoring are sufficiently important that they warrant independent discussion.

Risks result from a lack of knowledge or a lack of resources (e.g., time, money, people). Gaining knowledge or compensating for a lack of resources typically involves tailoring the process. For example,

design risks may be mitigated by gathering knowledge so that the risk in implementing the resulting design is minimized. Risks associated with customer/user acceptance may be mitigated by providing frequent interchange meetings. Schedule risks may be mitigated by architecting the process so that several activities are done in parallel (with the addition of mechanisms to ensure that the parallel effort does not introduce other risk). In fact, planning risk mitigation actions frequently results in process tailoring.

5.3 PROCESS TAILORING AND INCREMENT PLANNING

Increment planning is determining which process activities to include in each increment (see Section 4.3 for a discussion on increments). A development plan (e.g., *System Development Plan*) initially contains a definition of all the development increments at a high level. The process activities in the increments are not tailored, and the process may not be fully architected. Before implementing an increment, the process for that increment must be tailored. It is possible to tailor all the increments defined in the development plan when creating their initial definitions. However, it is not recommended to tailor the increments too far in advance of when they will be implemented because lessons learned and knowledge from implementing early increments should be used when tailoring later increments. Tailoring the increments prematurely may result in tailoring rework.

The first several increments are typically short and focused on reducing risk and gathering feedback. The results of the risk reduction and the implementation feedback are used to tailor later increments and create improved plans. As the development effort progresses, the increments get longer because many of the risks have been mitigated and the process and the plans are relatively stable.

5.4 APPLYING ISSEP ON A DEVELOPMENT EFFORT

The ISSEP model can be applied at any stage in a development effort, from initial concept development to final system test. This section describes how to apply the ISSEP model in two different points in the development life cycle: at the beginning and the middle of the life cycle.

It is difficult to clearly define when the life cycle begins. Does it begin with the arrival of the initial customer/user requirements, with the initial creation of the development plan, or even earlier, with the initiation of the preliminary discussions with the customer regarding the possibility of a development effort? The ISSEP model defines the start of the life cycle as the point in time when the initial development planning begins. The following discussion describes how to apply the ISSEP model from this point.

5.4.1 APPLYING ISSEP ON A NEW PROJECT

ISSEP tailoring begins with the execution of the management activities and creation of the initial version of the development plan. Creation of this plan is the first development increment. This long-term plan is based on the initial *Customer Needs* and the *Organizational Plans/Status*. The *Customer Needs* are baselined and become the initial contents of the first version of the development plan.

After each management activity, the information output from the activity is baselined and added to version 1 of the development plan. The context defined in the **Understand System Context** activity is the context for the entire development, and the risks identified and analyzed in **Analyze System Risks**

are the entire development risks. These activities are done at a high level, and the detailed context and risk analyses are done in future increments. The context and the risk analyses are input into the **Plan System Increment Development** activity and used to create a detailed plan to produce the initial development plan. In particular, this detailed plan describes the risk mitigation activities that must be performed to ensure that the initial development plan is realistic. After these mitigation activities are performed, the information is used to generate the initial development plan.

If the effort for performing the risk mitigation activities is sufficiently large, then it will take more than one increment (and perhaps several increments) to generate the development plan. In this case, the initial development plan describes risk mitigation activities and how the results of these activities will be used in developing successive versions of the plan. Eventually, sufficient risk mitigation is accomplished and incorporated into the version of the development plan that will be used to control the subsequent development activities.

After version 1 of the development plan is created (even if it only contains increments for doing risk mitigation activities), the plan is baselined as part of the *System Baseline/Plan/Status*. When changes are made to any part of the *System Baseline/Plan/Status* (i.e., a new set of *Customer Needs* is input, the results of the development effort are added to the baseline, status information is added, or changes are made to the development plan), a new version of the baseline is established and all parts of the new version are appropriately examined and updated. This examination and updating is done in the management activities. Consequently, every time information is input into the management activities a new version of the *System Baseline/Plan/Status* is created. This ensures that the development effort remains synchronized; that is, the plans always reflect the current status and are based on the current development information, and the development work is being controlled by the latest-and-greatest version of the development plans.

5.4.2 Applying ISSEP to an Ongoing Project

An ongoing project is defined as one that already has some form of a development plan and is at some point in the process of implementing that plan. This section defines the type of ongoing projects that would benefit most from implementing the ISSEP model and describes how the model would be applied in this case.

One of the main objectives of the ISSEP model is to help development efforts maintain control. ISSEP feedback loops encourage the early detection and resolution of problems and help projects avoid making overly optimistic projections. Although applying a process like ISSEP is beneficial from the outset of a development effort, it is also appropriate to use the process on an ongoing effort. Although ISSEP could be applied to any ongoing effort, ISSEP's control mechanisms make it the process of choice for efforts that are "out of control" because they provide a means to regain project control. The ISSEP model accomplishes this by establishing activities and information links that gain control while requiring only the minimum necessary rework and not requiring the effort to effectively "start over" with all new work products.

The first step in applying ISSEP to ongoing projects is to implement the management activities. Although this is the same first step as when implementing ISSEP on a new project, the implementation is slightly differently for ongoing projects. The initial *System Baseline/Plan/Status* that is baselined includes all previous plans, status information, and the current version of the system parts that have been developed as well as the *Customer Needs*. The high-level context and risks that are identified and

analyzed are based on the project's current state. All assumptions are noted and considered during the risk analysis; however, rework is not initiated until other less costly approaches have been eliminated. The plan that is created in this first ISSEP increment (or several increments, if necessary) defines the plan for continuing the development effort. Although the project is not just beginning, the first several increments after initiating ISSEP are short. In these increments, risks are mitigated and the plan is refined.

The ISSEP feedback mechanisms ensure that any errors in the developed work products, or any risks associated with the continued effort, surface as the development work continues. For example, the development plan for the ongoing project may begin with a validation and verification of the design at all system levels. If no verified requirements specification is available at a level, the requirements specification will need to be created. If the specification that exists has not been verified, it will be required to be verified. If the verification uncovers defects in the specification, the specification will need to be modified, and so on. Each of these activity sets may be in separate increments (and the development plan may be modified after each), or the plan may indicate that within certain constraints, the work can be done within the same increment. The decision about what is included in an increment is based on the need for frequent feedback.

After several increments, the plan becomes more realistic, and the ISSEP information flows are reasonably well established. At this point, any differences in the process based on when the process was introduced are insignificant.

5.5 USING ISSEP WITH DIFFERENT LIFE-CYCLE MODELS

The ISSEP model is independent of life cycle and can be used to implement any life-cycle model such as waterfall, incremental, and evolutionary. This independence is because the ISSEP model defines how work will be done, and a life-cycle model defines the stages through which the developed products transition on their way to completion. Although there are similarities between development models like ISSEP and life-cycle models (e.g., you do design and create the design for the system), the main difference is that the life-cycle model defines a specific ordering of product stages. The development model is not order-dependent. Therefore, the ISSEP model does not define order for creation of system parts or determine whether the parts are developed in multiple builds or the entire system is created in a single build.

The following discussion explains how to select a life-cycle model and instantiate the ISSEP model for that life cycle. Sections 5.5.1 through 5.5.3 describe how to use the ISSEP model when implementing the three life-cycle models mentioned above.

In the ISSEP model, the initial version of the development plan contains a description of the life-cycle model that will be followed during the development. The model selection is made in the **Develop/Update System Plan** activity. The main criteria for selecting the life-cycle model are contractual delivery requirements, the amount of interfacing necessary to establish complete and accurate system requirements, and the system risk profile (i.e., whether the system is a high or low risk development effort). An analysis is done to determine which type of model or combination of models will be most appropriate. The process tailoring and eventual instantiation of the ISSEP model is greatly influenced by the life-cycle model chosen. If the project situation changes and the life cycle must be

changed or modified, some process tailoring rework will be necessary; but, because the ISSEP model is not life-cycle dependent, the development process itself can remain unchanged.

5.5.1 ISSEP AND THE WATERFALL LIFE-CYCLE MODEL

The waterfall model, traditionally used for development of software systems, defines a sequential set of activities. Each activity is implemented, and the product(s) developed during implementation are verified. If the products pass verification, the next activity in the sequence begins. If the verification failed, the activity is reimplemented and reverified. If a defect is identified that forces a previous activity's work products to be modified, control is passed back to the correcting activity; from there, the initial sequence is repeated. When implementing the waterfall model, the entire system goes from activity to activity as a whole.

The waterfall model is most appropriate when the requirements are well defined at the onset of development and the project is low risk. This situation implies that there is less probability that defects will be found late in the life cycle, which will necessitate rework. The waterfall model is less appropriate if not all parts of the system will spend the same amount of time in each activity because time will be wasted waiting for the system parts that take more time to complete an activity.

If the waterfall model is selected when the initial development plan is created, the ISSEP model is tailored such that each of the waterfall activities maps to an ISSEP increment. For example, the first increment after the development plan is complete might be the first waterfall activity, the next increment the next waterfall activity, and so on. When applying the waterfall model with ISSEP, a simple tailoring might be to implement a risk reduction waterfall, where the risks identified by the ISSEP activities are used to determine the rigor that is necessary in subsequent activities. The risk reduction approach helps minimize the rework that can result when defects are found late in the life cycle.

5.5.2 ISSEP AND THE INCREMENTAL LIFE-CYCLE MODEL

"The 'incremental' strategy determines user needs and defines the system requirements, then performs the rest of the development in a sequence of builds. The first build incorporates part of the planned capabilities, the next build adds more capabilities, and so on, until the system is complete" (Department of Defense 1994). After the requirements are defined, the builds can follow any appropriate process for development.

The incremental model is most appropriate when the requirements are initially well defined, and they can be grouped into increments that can be developed independently without resulting in undue integration risk.

If the incremental model is selected when the initial development plan is created, the ISSEP model is tailored such that the next increment analyzes the requirements for the entire system using the ISSEP model's iterative approach to gather information from each of the system parts in the decomposed system. The builds are defined during implementation of the next set of management activities, are documented in the next version of the development plan, and are based on the requirements analysis. The defined builds are mapped to ISSEP increments such that the next set of increments produces the first system build, the next set produces the second system build, and so on. The increments can be defined to allow the builds to overlap so that as the first build completes design and begins implementation, the next build begins design.

5.5.3 ISSEP AND THE EVOLUTIONARY LIFE-CYCLE MODEL

"The 'evolutionary' strategy also develops a system in builds, but differs from the incremental strategy in acknowledging that the user need is not fully understood and all requirements cannot be defined up front. In this strategy, user needs and system requirements are partially defined up front, then are refined in each succeeding build" (Department of Defense 1994). Similarly to the incremental model, each build may follow any development process.

The evolutionary life-cycle model is most appropriate when the customer requirements are not well understood initially, and feedback from the customer is necessary to ensure a successful development.

If the evolutionary model is selected, the initial development plan describes the approach for determining the requirements for the initial build. The next increment implements that approach and defines the requirements for the first build. Then the management activities are implemented, and the next version of the development plan is created to define the initial build and map it to ISSEP increments. Either the entire build can be complete before the customer provides feedback, or the customer can be involved at other points in the process and provide more frequent direction. The customer feedback points and their focus are defined in the development plan.

Although the ISSEP model can work well with all these life-cycle models, its real strengths are evident when implementing ISSEP on incremental or evolutionary efforts because the ISSEP model's communications mechanisms and risk-based approach provide the extra information needed when the development is high risk or the requirements are incomplete or unstable.

5.6 EXAMPLE APPLICATION OF THE ISSEP MODEL

The radar subsystem in Figure 15 is an example of applying the ISSEP model to a system that is decomposed into multiple levels.

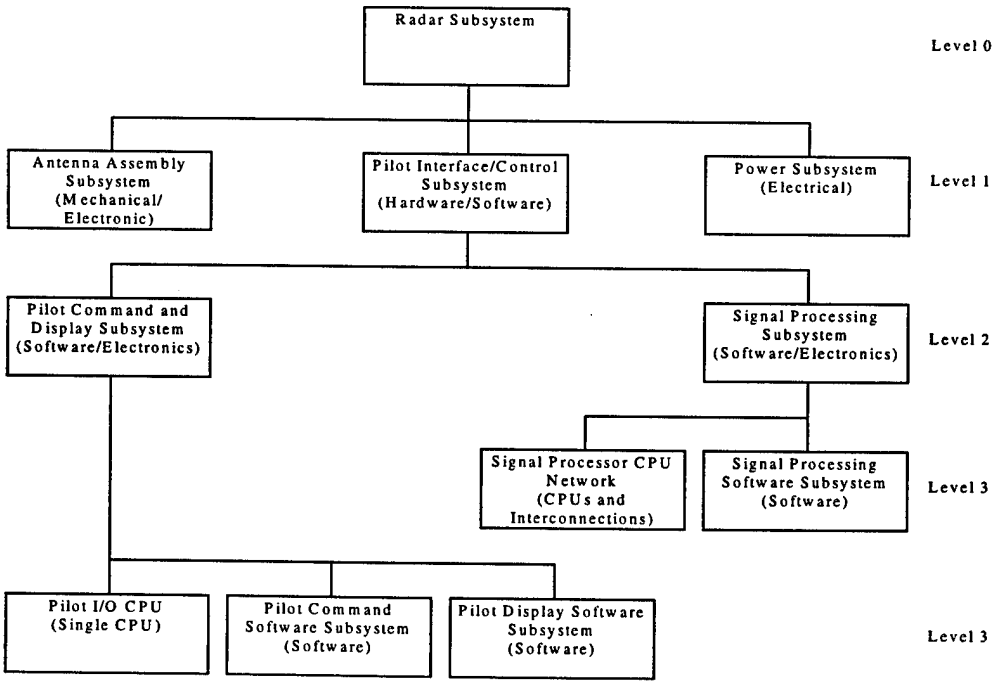


Figure 15. Partial Decomposition of a Radar Subsystem

The ISSEP process for the radar subsystem would include the following:

Develop Level (0) (Radar Subsystem)

Develop Level (1) (Antenna Assembly Subsystem)

Develop Level (1) (Pilot Interface/Control Subsystem)

Develop Level (2) (Pilot Command and Display Subsystem)

Develop Component (Pilot CPU)

Develop Level (3) (Pilot Command Software Subsystem)

Develop Level (3) (Pilot Display Software Subsystem)

Develop Level (2) (Signal Processing Subsystem)

Develop Level (3) (Signal Processor CPU Network)

Develop Level (3) (Signal Processor Software)

Develop Level (1) (Power Subsystem)

The activities for each instance of the process at each level are identical except that the design activities can be system design activities or CI (i.e., software or hardware) design activities.

To understand the relationship between levels, consider the roles of a system or software engineer. The essence of system and software engineering consists of refining and allocating customer requirements to subsystems; allocating design decisions to subsystems; and ensuring that, once designed, the subsystems work together properly. At lower levels of the process, requirements and design decisions can also be allocated to individual components.

Requirements can be broadly classified as functional and nonfunctional. Functional requirements describe how an element of the system must behave. A functional requirement describes how an element should consume inputs and transform them into outputs. For example, a functional requirement to highlight the image of an aircraft that is too close to the host aircraft could be imposed on the entire radar subsystem. This requirement could be divided among the subsystems as follows:

- The Signal Processing Software Subsystem would identify specific images as aircraft and tag each image with the distance from the host.
- The Pilot Command Software Subsystem would allow the pilot to define a critical distance.
- The Pilot Display Software Subsystem would use the critical distance to select aircraft images for highlighting.

Nonfunctional requirements specify constraints on how the functional requirements can be implemented. An example of a nonfunctional requirement is the upper bound on the time for the pilot display to reflect the change in position of another aircraft. This nonfunctional requirement would be imposed on the entire radar subsystem and divided among the following subsystems:

- Antenna Assembly Subsystem (influences rotational speed of the antenna)
- Pilot Interface/Control Subsystem (influences the lower bound on CPU speed and the upper bound on software stimulus response time)
  - Pilot Command and Display Subsystem (influences the upper bound on stimulus response time of the two software subsystems and the lower bound on CPU speed)
    - — Pilot Command Software Subsystem (influences the upper bound on software stimulus response time)
    - — Pilot Display Software Subsystem (influences the upper bound on software stimulus response time)
  - Signal Processing Subsystem (influences the lower bound on the speed of CPUs and network and the upper bound on software stimulus response time)
    - — Signal Processor CPU Network (influences the lower bound on speed of CPUs and network)
    - — Signal Processor Software (influences the upper bound on software stimulus response time)

In addition to allocating customer requirements, design decisions will impose additional requirements on subsystems and components. For example, partitioning of requirements between the Pilot Interface/Control Subsystem and the Antenna Assembly Subsystem necessitates a physical interface between the two. Interface details (e.g., use of an electrical rather than a fiber-optic connection) would be specified as part of Design and Verify Level (0). This design decision would flow down as a physical interface requirement through the Pilot Interface and Control Subsystem to the Signal Processing Subsystem. Finally, in Design and Verify Level (3) for the Signal Processor CPU Network, the interface between the electrical cable and the CPU network would be specified. The characteristics of this interface would impose requirements on design of the CPU network and the Signal Processing Software Subsystem.

As another example, consider the Pilot Command and Pilot Display Software Subsystems. To implement a zoom and roam requirement, the Design and Verify Level (2) activity for the Pilot Command and Display Subsystem could decide the software function in the Pilot Command Software subsystem that would have to synchronize with a function in the Pilot Display Software Subsystem. This decision would impose an interface requirement on the Develop Level (3) processes for these two subsystems. It could also affect selection of an operating system, which would impose additional requirements on the two Develop Level (3) processes. (It is assumed that both software subsystems run on the Pilot CPU.)

The set of functional and nonfunctional requirements imposed on a component or subsystem can be viewed as a contract between levels of the process. In addition to requirements on the item under development, process requirements (such as for periodic reviews and status reporting) are part of the contract between levels. For example, the Level (2) *Development Plan/Status* input to the Signal Processor Software Subsystem could require status reporting on a biweekly basis and upon completion of specific milestones, such as completion of the architectural design.

When implementing the ISSEP model, it is necessary to pass design issues to the next higher level for resolution. For example, the team developing the Pilot Display Software Subsystem could decide that the subsystem has too little time to highlight an aircraft that is too close to the host because it must compare the distance tag for each aircraft with the critical distance provided by the pilot. The risk could be mitigated in at least two ways:

1. The software could be provided with its own CPU, or the CPU it shares with the other Pilot Command Software Subsystem could be upgraded. This decision could be made by the process for the Pilot Command and Display Subsystem. It would require higher-level buy-in if it impacted cost constraints or other requirements.

2. The Signal Processing Software Subsystem could be given the responsibility of determining which aircraft are too close to the host. Providing this ability would involve adding a function to that subsystem and adding a logical interface between it and the Pilot Command Software Subsystem (which provides the critical distance). The decision would have to be a consensus among development processes for these two subsystems, the two Level 2 processes in Figure 15, and the Pilot Interface and Control Subsystem process at Level 1. Buy-in from the Signal Processor CPU Network process might be required as well.

The ISSEP model also requires buy-in on decisions from the next higher level. There are several kinds of examples of this decision. One is refinement of an ambiguous requirement. A typical example is specifying details of subsystem-to-subsystem interfaces. Consider the interface between the Signal Processing Software Subsystem and the Pilot Display Software Subsystem. This situation is almost identical with the one described in Option 2 of the previous example. That is, the development teams for these two subsystems could work out the details informally, but their decisions would need approval from, at minimum, the processes for the Signal Processing, Pilot Interface/Control, and Pilot Command and Display Subsystems. Design of an interface between the Pilot Command and Pilot Display Software Subsystems, on the other hand, could require approval only from the and Pilot Command and Display Subsystem process.

Another example is in the area of detailed user interface requirements. Requirements passed to the Pilot Display Software Subsystem process might state only that the display format shall enable the pilot to determine, in less than 1 second, whether the host aircraft is in imminent danger of colliding with another aircraft. If the requirement was this vague, the Pilot Display Subsystem process would have ensured that its elaboration had buy-in from all stakeholders. This could be accomplished via direct communication between that process and the stakeholders or by assigning joint responsibility to that process and one or more ancestor processes.

Another type of decision that needs higher-level approval is the decision to change a tradeoff between nonfunctional requirements. For example, the Signal Processing Software subsystem might determine that it could greatly increase the accuracy of aircraft position if it could be allowed a little more computation time. This change would require buy-in from higher levels, because it would probably necessitate taking time away from another subsystem or changing the overall timing requirements for the Radar Subsystem. This example is a specific case of changing requirements allocation, a decision which requires approval from the activity that made the allocation in the first place. In addition to changing allocation of nonfunctional requirements, functional requirements can be moved from one subsystem to another.

This page intentionally left blank.

6. PROCESS ISSUES REVISITED

The ISSEP model is intended to be useful for real-world systems and software engineering. To ensure practicality, the ISSEP model addresses the key systems and software engineering process issues and is compatible with the standards most likely to be imposed on system and software development efforts. This section revisits the key process issues described in Section 2 and shows how the ISSEP model resolves them. The key process issues are:

- Integrated management and technical activities
- Standards compliance
- Managing complexity
- Process adaptability and tailorability

6.1 INTEGRATED MANAGEMENT AND TECHNICAL ACTIVITIES

The ISSEP model defines process interfaces between the management and technical activities and between the systems and software development activities. These interfaces define the information exchange mechanisms that integrate these sets of activities. The information exchange mechanisms include the information flows (i.e., the specific information that is passed across the interface) and the activity descriptions that define how the information is used for generating new information and determining where information should be routed.

Much of this technical report describes the information exchange mechanisms. The management activities rely on the technical activities to provide the information they need for decision making. The technical activities rely on the management activities for creating the baselines and developing the plans that control the implementation. The systems-level activities rely on information from the software activities to ensure that the designs can be developed with minimal risk and that they can be efficiently integrated once implemented. The software activities rely on the system-level activities to provide the high-level design and integration plans that scope and direct the efforts. Without all the pieces, the ISSEP process would be missing a vital part of what makes the process work. In fact, the management and technical and systems and software activities are so tightly integrated, that it would be awkward to separate any part from the whole and still have an implementable process model.

6.2 STANDARDS COMPLIANCE

The approach for development of the ISSEP model has built on existing work done in systems and software engineering. Specifically, the processes and work-product descriptions in several well-known standards have been reviewed, and the ISSEP model is compliant with them. The specific standards reviewed during ISSEP development were MIL-STD-498, EIA/IS-632, P1220, ISO/IEC 12207, the SE-CMM, and the CMM. These standards describe required systems and software development activities and may additionally describe the work products produced by those activities. The standards tend not to define the information flow between activities.

The standards with which the ISSEP model complies are largely software engineering or systems engineering standards. They concentrate on the management or technical activities and do not define specific information exchange mechanisms between them. Although these standards are extremely useful and provide substantial guidance, their focus on one aspect of development of large, complex systems means that much of the task of implementing the different standards concurrently is left to the implementors. The ISSEP model provides a means to effectively combine the standards. Following an ISSEP generated process, developers have the ability to select the best aspects of several standards.

The ISSEP model defines activities and work products so that the requirements documented in the standards can be implemented. The ISSEP model focuses on high-level process activities and the information flows that contribute to successful implementation and provides a framework for application of these standards.

The ISSEP model's emphasis on information helps users of standards focus on the engineering, rather than on document production. During process tailoring, the documents required by the standards can be directly mapped to the ISSEP defined information, and a documentation task can be specified that produces the document(s) from the already available information. In this way, the development process is truly an engineering process where document generation is a by-product of the effort, not the focus.

ISSEP model compliance does not mean that every activity in every standard can be directly mapped to an ISSEP activity, although many can. However, the ISSEP model does not preclude the inclusion of any of the standard activities. Because the ISSEP model is at a high level, most of the development activities in the standards are directly mappable, but the mapping is not one-to-one. This technical report concentrated on the software standards, and Appendix F defines a mapping of the ISSEP model to the software engineering standard MIL-STD-498 and the CMM. The predecessor to this report, *A Tailorable Process for Systems Engineering* (Software Productivity Consortium 1995b), which included the ISSEP systems engineering and management activities, provides a mapping to the systems engineering standards EIA-IS-632 and the SE-CMM.

6.3 MANAGING COMPLEXITY

There are two ways that the ISSEP model manages complexity. First, the appropriate levels of abstraction for the development of each system part are defined. Second, realistic communication mechanisms that reduce the effects of fragmented development are defined.

Level of Abstraction. This report describes how the ISSEP model is decomposed to mirror the decomposition of the developing system. However, if the highest level processes had to "solve the entire

problem" in detail in order to control the lower level activities, the problem would still be complex, and the benefits of the decomposition would not be realized. Rather, in the ISSEP model, the system parts at each level define the system at an appropriate level of abstraction depending on their level in the hierarchy. That is, the higher level activities develop designs and plans at a high level and lower level activities develop designs and plans in more detail. By allowing each level to remain focused within their level of abstraction, the job of dealing with the complexity of the entire system is distributed among all the parts. This is an effective way of managing complexity if communications mechanisms are in place to transfer information.

Communication Mechanisms. The ISSEP model defines communication mechanisms that promote the flow of information. This means that information defined by the model is routinely communicated. For example, any risks identified by the design activity are transferred to the management activity for further analysis. The management activity does not have to request the design activity to transfer the risks. On the other hand, in process models, where communication mechanisms do not promote the flow of information, information must be actively requested before it is available. The ISSEP model's communication mechanisms promote information exchange throughout the system hierarchy. The information exchange mechanism helps manage complexity because it reduces the impact of fragmentation caused by the distributed development.

The ISSEP communication mechanisms also encourage all participants in the development to take a proactive role in the successful completion of the system. The more knowledgeable the participants are of the development effort, the more likely they are to feel a part of a team effort, and the more likely they are to find and resolve defects. The psychological factors of a strong team, empowered with knowledge, working together to produce a complex system, cannot be overlooked. The ISSEP communication mechanisms provide the foundation for this approach which, if implemented, can make a very large team work as effectively as a small one.

6.4 PROCESS ADAPTABILITY AND TAILORABILITY

The ISSEP model is adaptable and can be tailored and instantiated to generate system and software development processes for projects of any size, complexity, architecture, domain, or organizational structure. As discussed in Section 5.5, the ISSEP model can be applied with any life-cycle model. These features make ISSEP an attractive process model for organizations that develop diverse types of projects or need a process that can easily fit into their current culture. The ISSEP model can be introduced into an organization without making changes to many of the current subprocesses such as organizational processes (e.g., reporting channels), document production, peer review, training, quality assurance, and configuration management. In fact, because the ISSEP model is high-level, organizations that already have an organizational standard development process can still adopt the ISSEP model as a process framework of which their current process can be a part.

As discussed in Section 5.1, the ISSEP model, like all process models, must be tailored for use by a project. The ISSEP model helps tailoring by defining the activities that perform the tailoring: **Plan System Increment Development** and **Plan CI Increment Development.** The ISSEP model assumes that tailoring is part of project planning, thus, these activities must identify the information that is considered when performing the tailoring.

Process tailoring and process adaptability complement each other. If a process is at a sufficiently high level to be easily adaptable, the process is high level enough that it needs a rigorous amount of process tailoring to make it enactable. Although high-level processes require more tailoring, their flexibility to accommodate a wide variety of development strategies makes them appropriate choices for adoption. By providing tailoring guidance, the ISSEP model facilitates process tailoring and compensates for the additional rigor needed due to its generic nature.

A. ISSEP IDEF0 DIAGRAMS

This appendix contains the IDEF0 diagrams for the ISSEP model. Section 3.2.1 contains a description of the IDEF0 notation.

Table 5. Diagram Structure for Appendix A

Diagram Number	Diagram Title	Section 3 Figure Number	Appendix A Figure /Page Number
A-0	Context	Figure 4	Figure 16, Page 60
A0	Develop Operational System	Figure 5	Figure 17, Page 61
A1	Manage System Development	Figure 6	Figure 18, Page 62
A2	Design and Verify System	Figure 7	Figure 19, Page 63
A3	Develop Configuration Item	Figure 8	Figure 20, Page 64
A31	Manage CI Development	Figure 9	Figure 21, Page 65
A32	Design and Verify CI	Figure 10	Figure 22, Page 66
A33	Develop Component	Figure 11	Figure 23, Page 67
A34	Integrate and Test CI	Figure 12	Figure 24, Page 68
A4	Integrate and Test System	Figure 13	Figure 25, Page 69

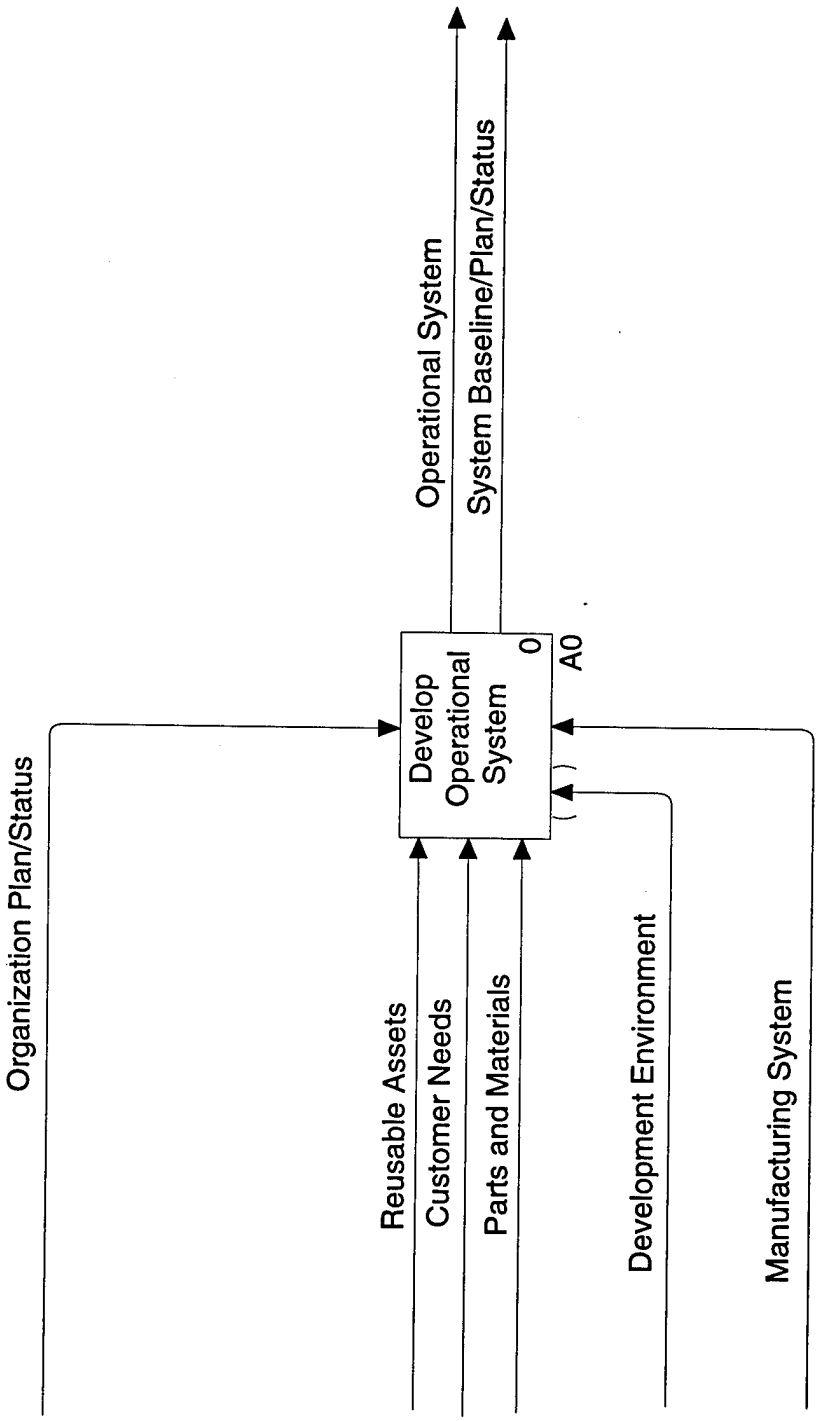


Figure 16. Context (A-0)

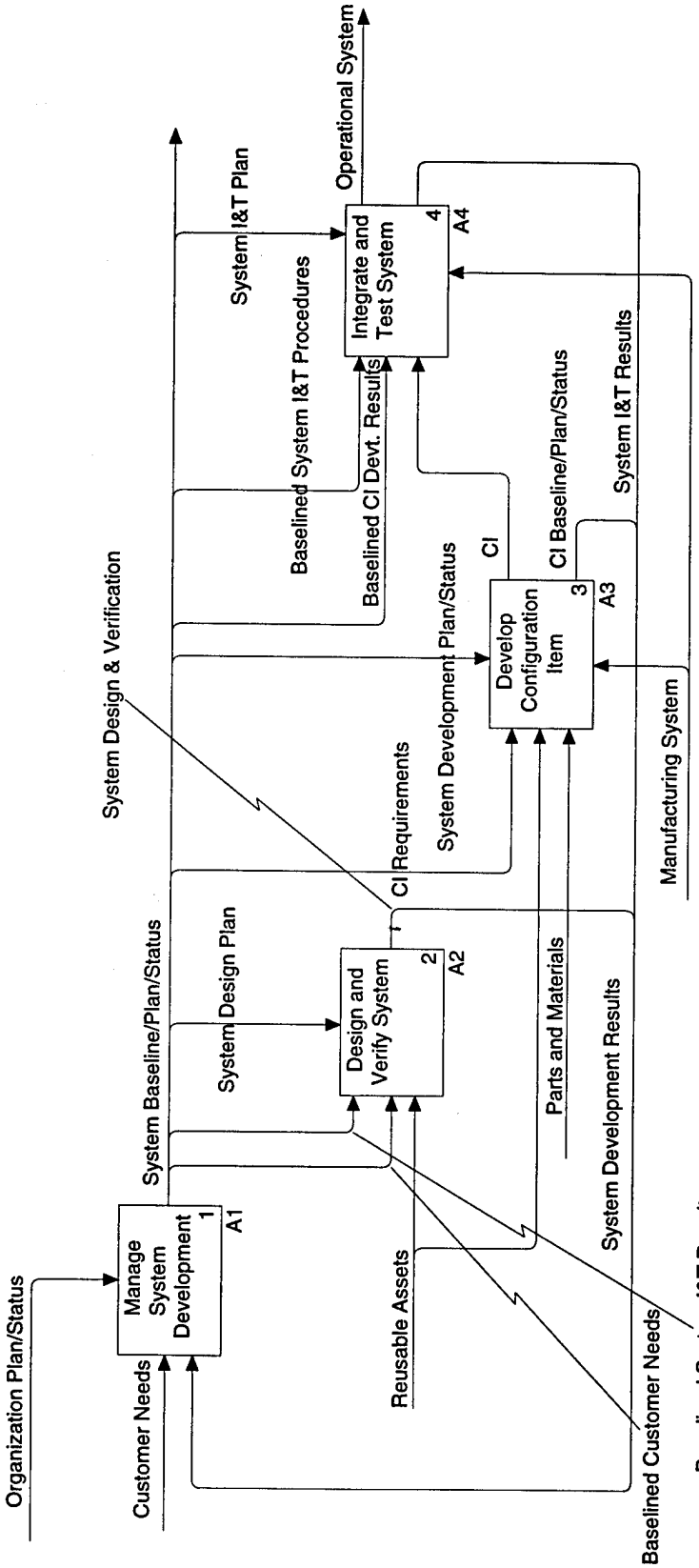


Figure 17. Develop Operational System (A0)

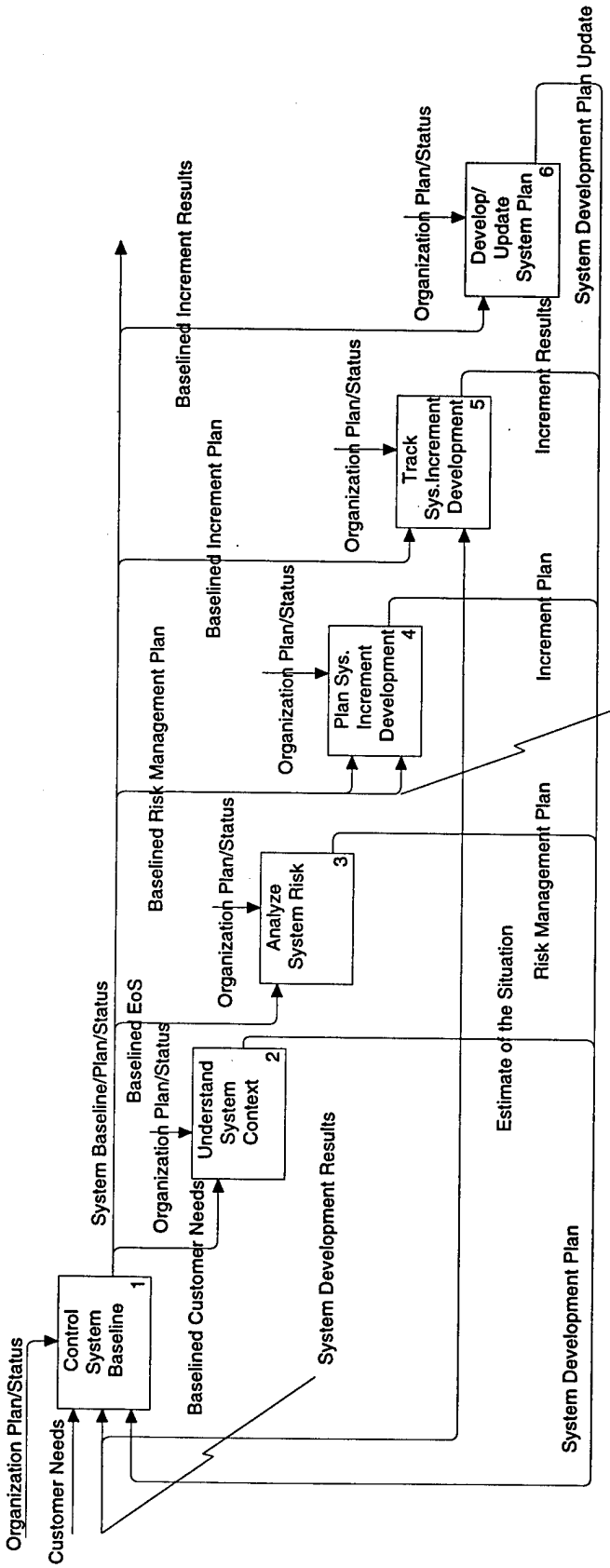


Figure 18. Manage System Development (A1)

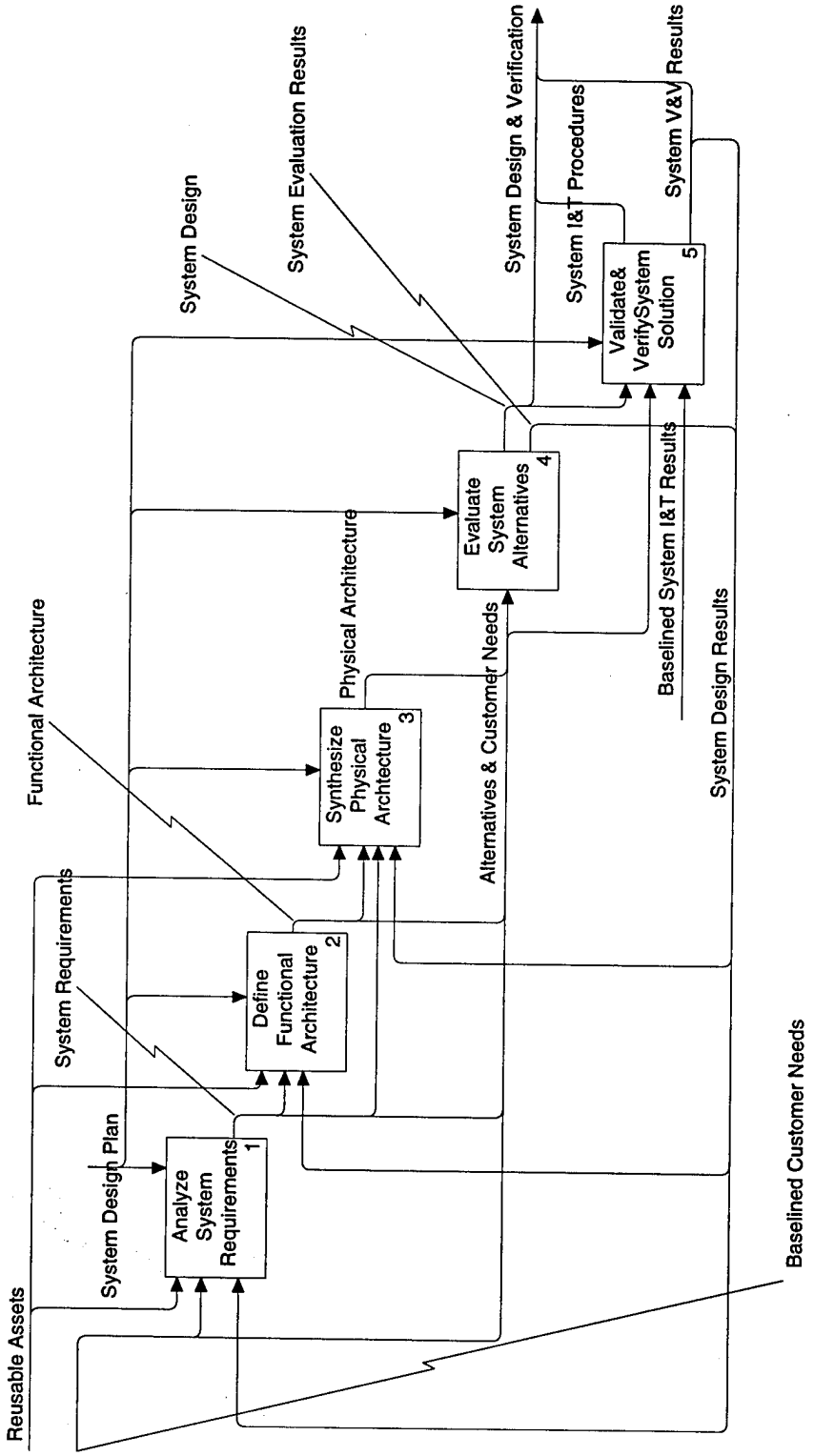


Figure 19. Design and Verify System (A2)

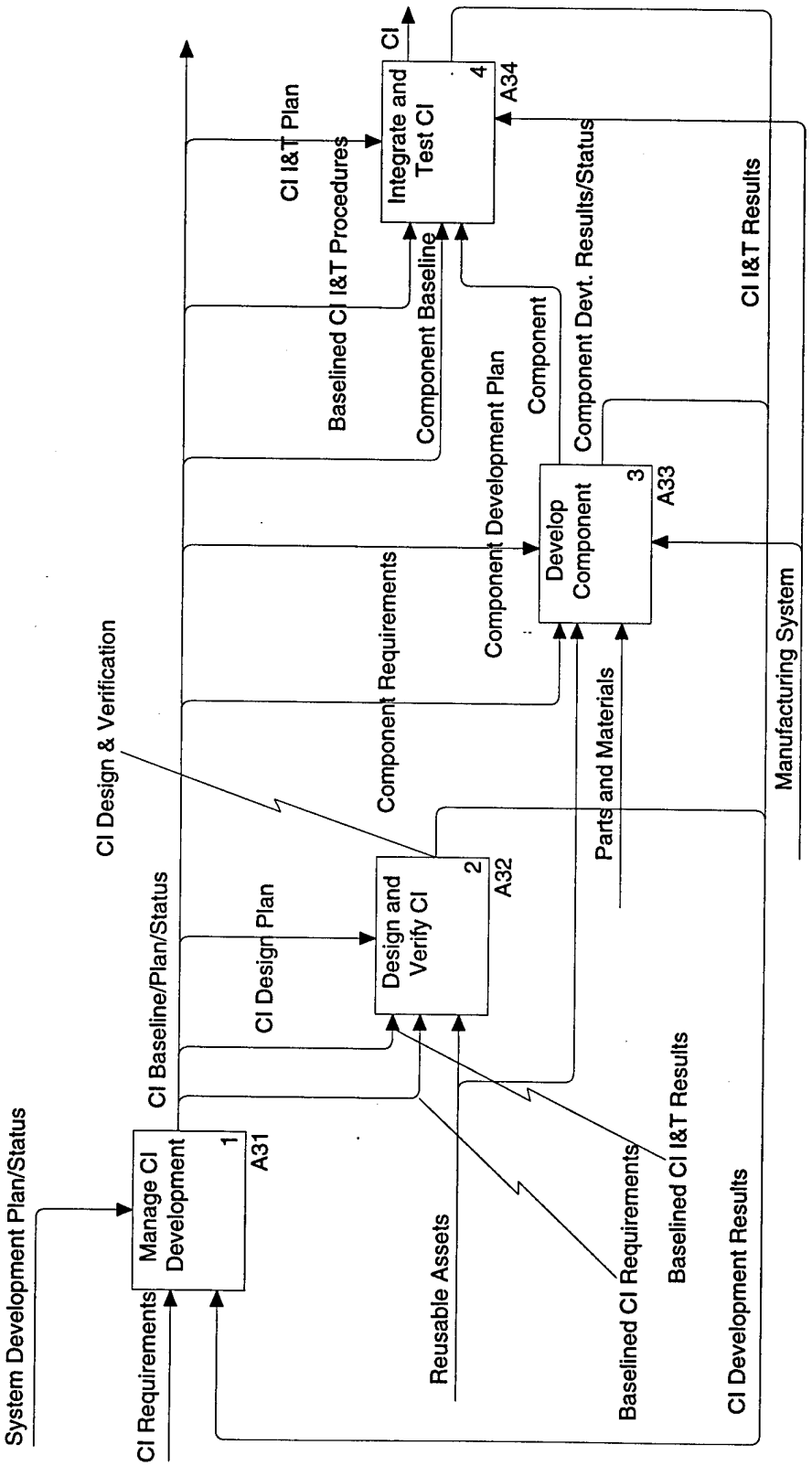


Figure 20. Develop Configuration Item (A3)

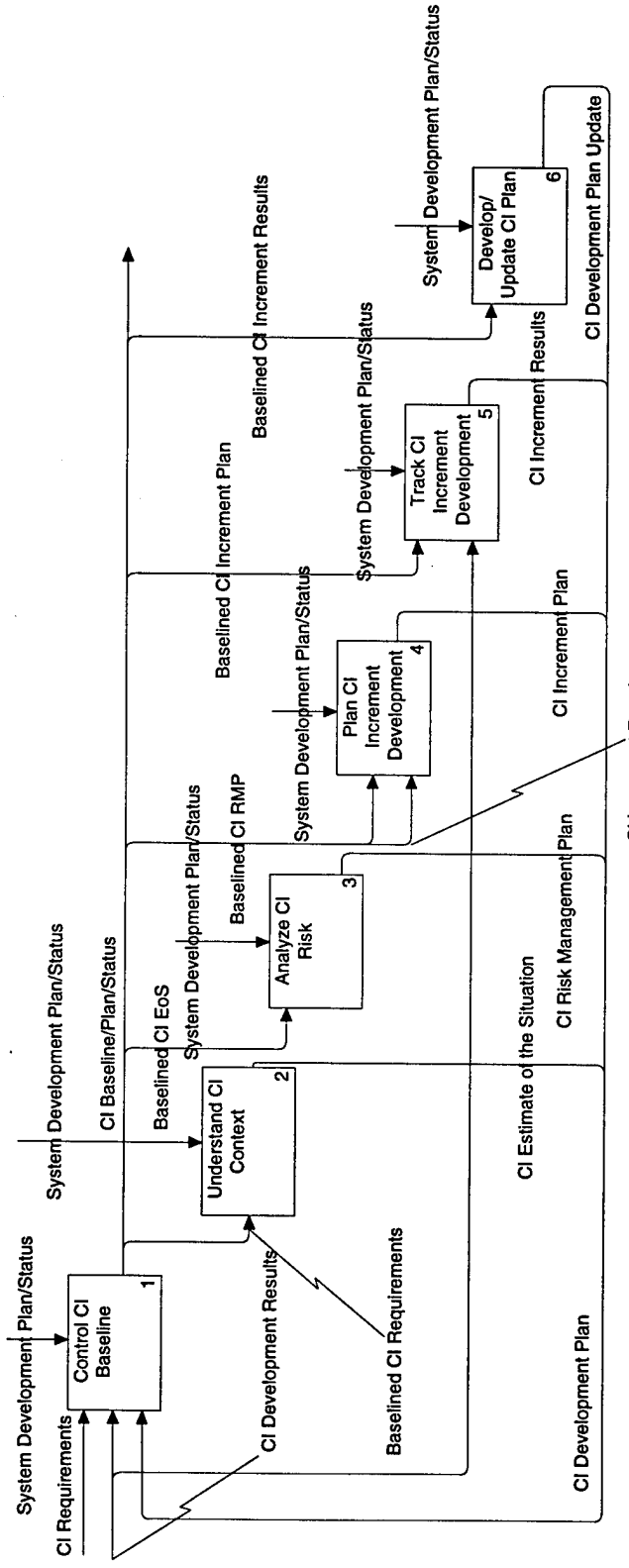


Figure 21. Manage CI Development (A31)

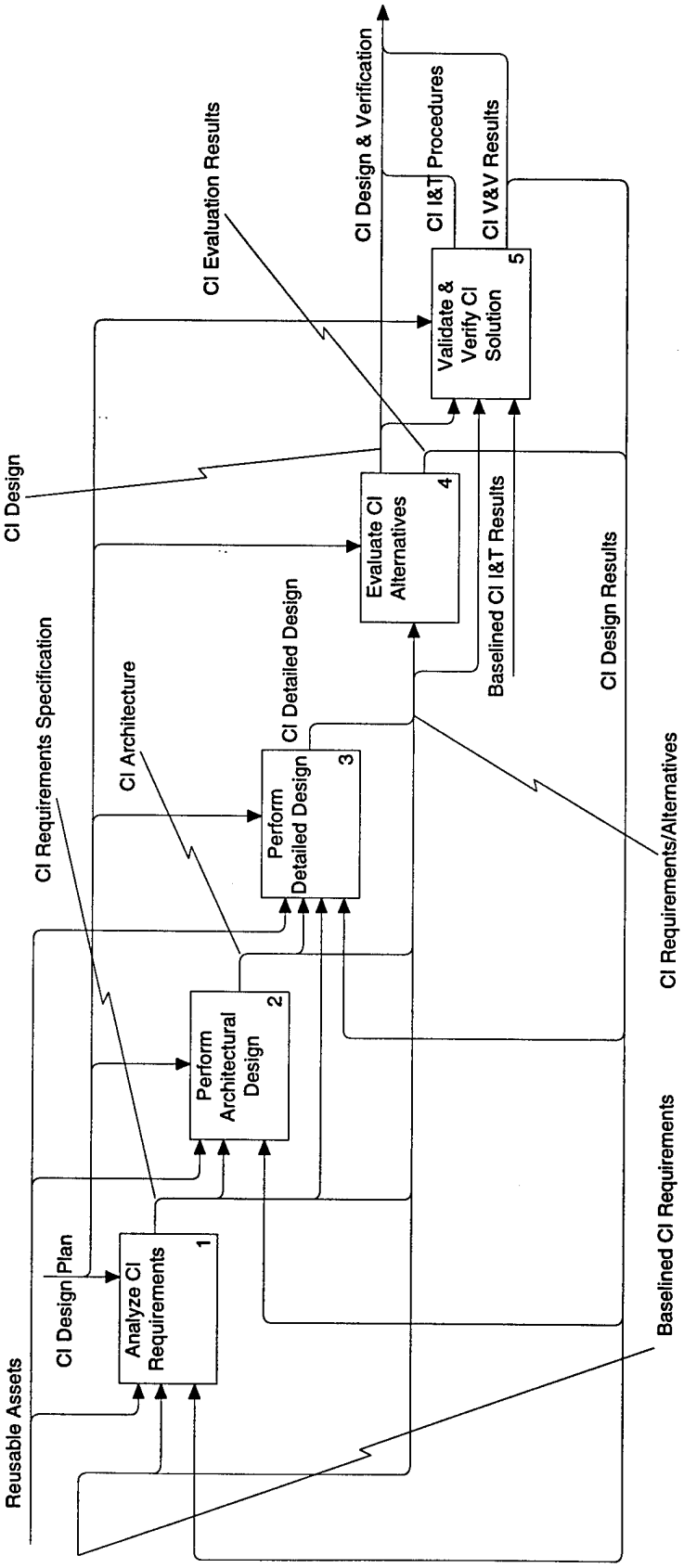


Figure 22. Design and Verify CI (A32)

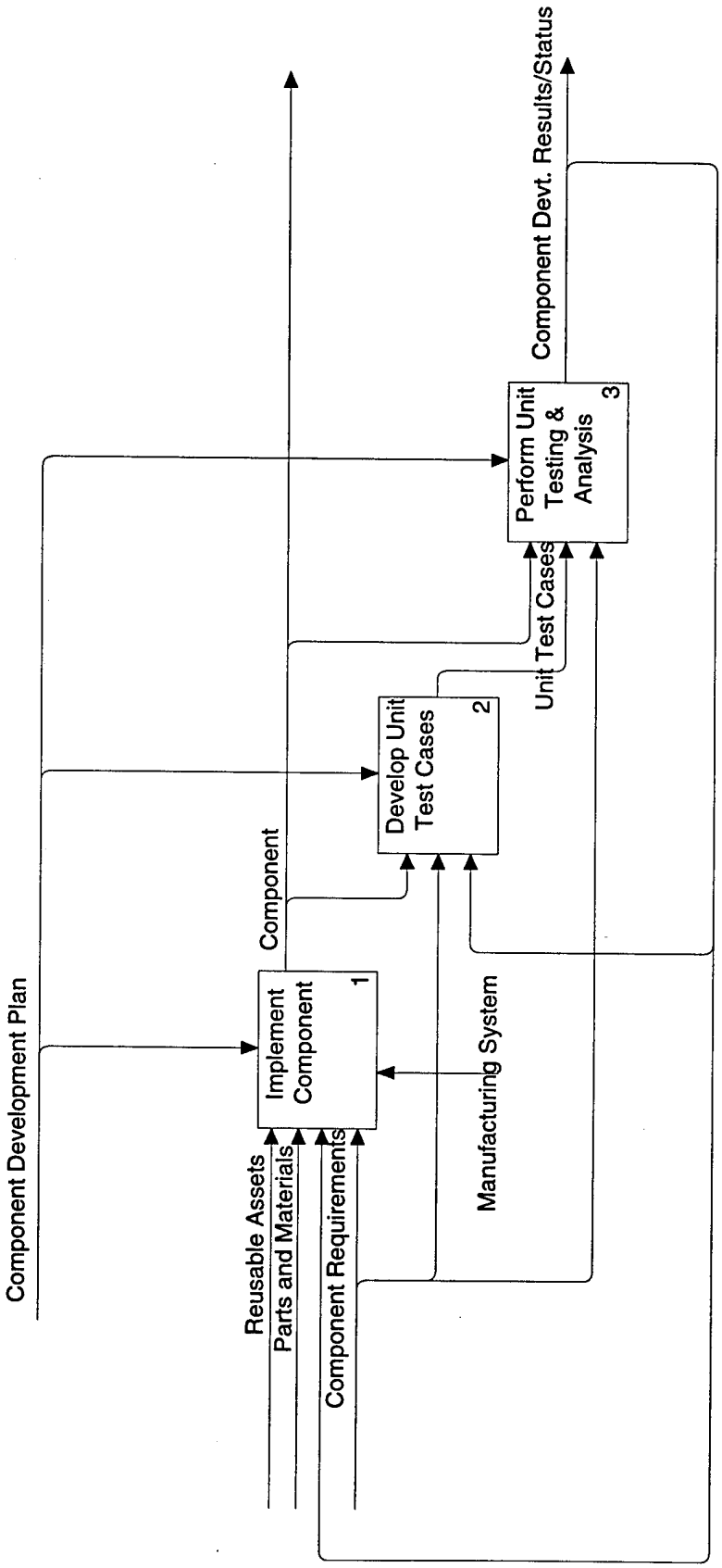


Figure 23. Develop Component (A33)

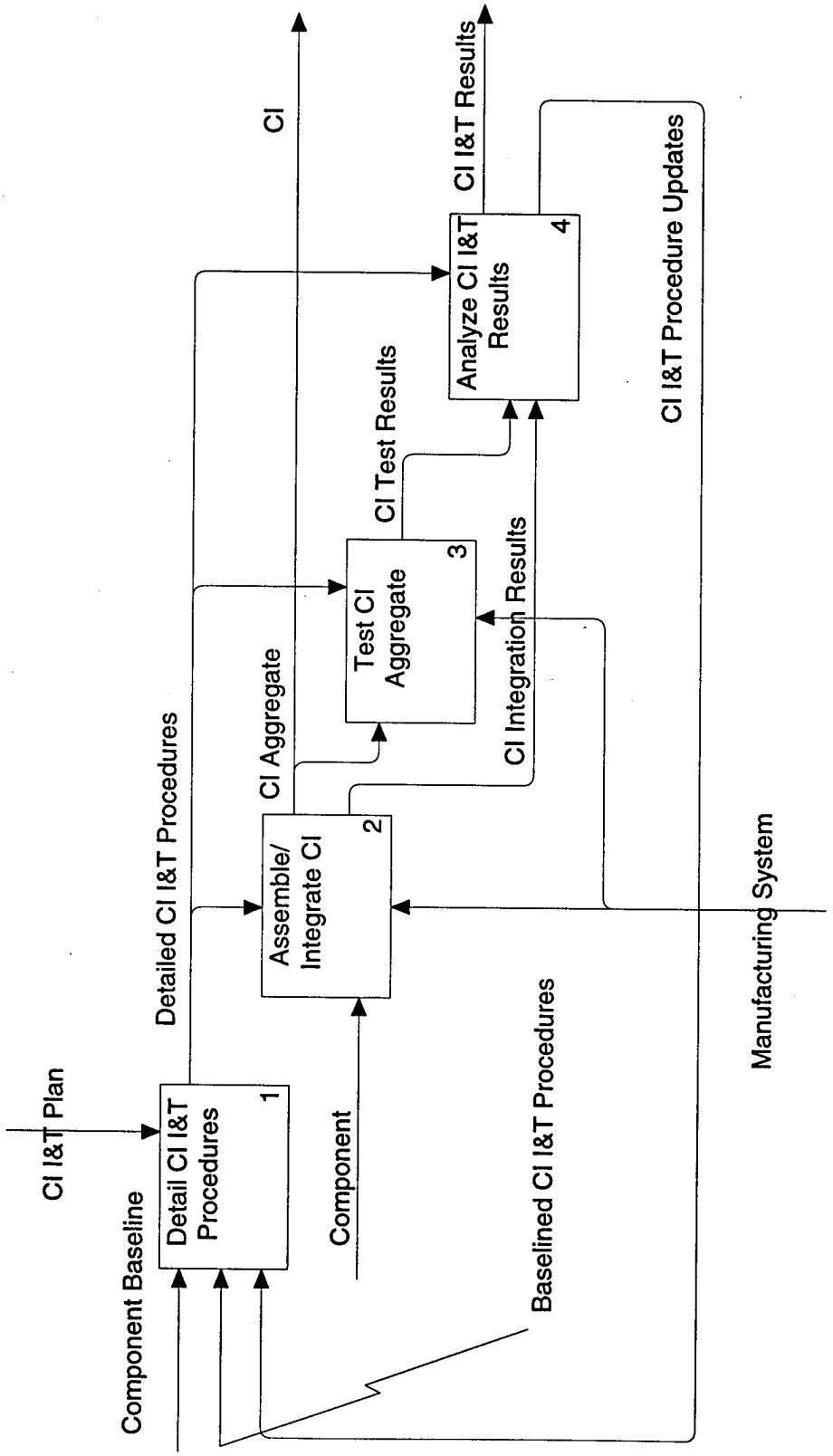


Figure 24. Integrate and Test CI (A34)

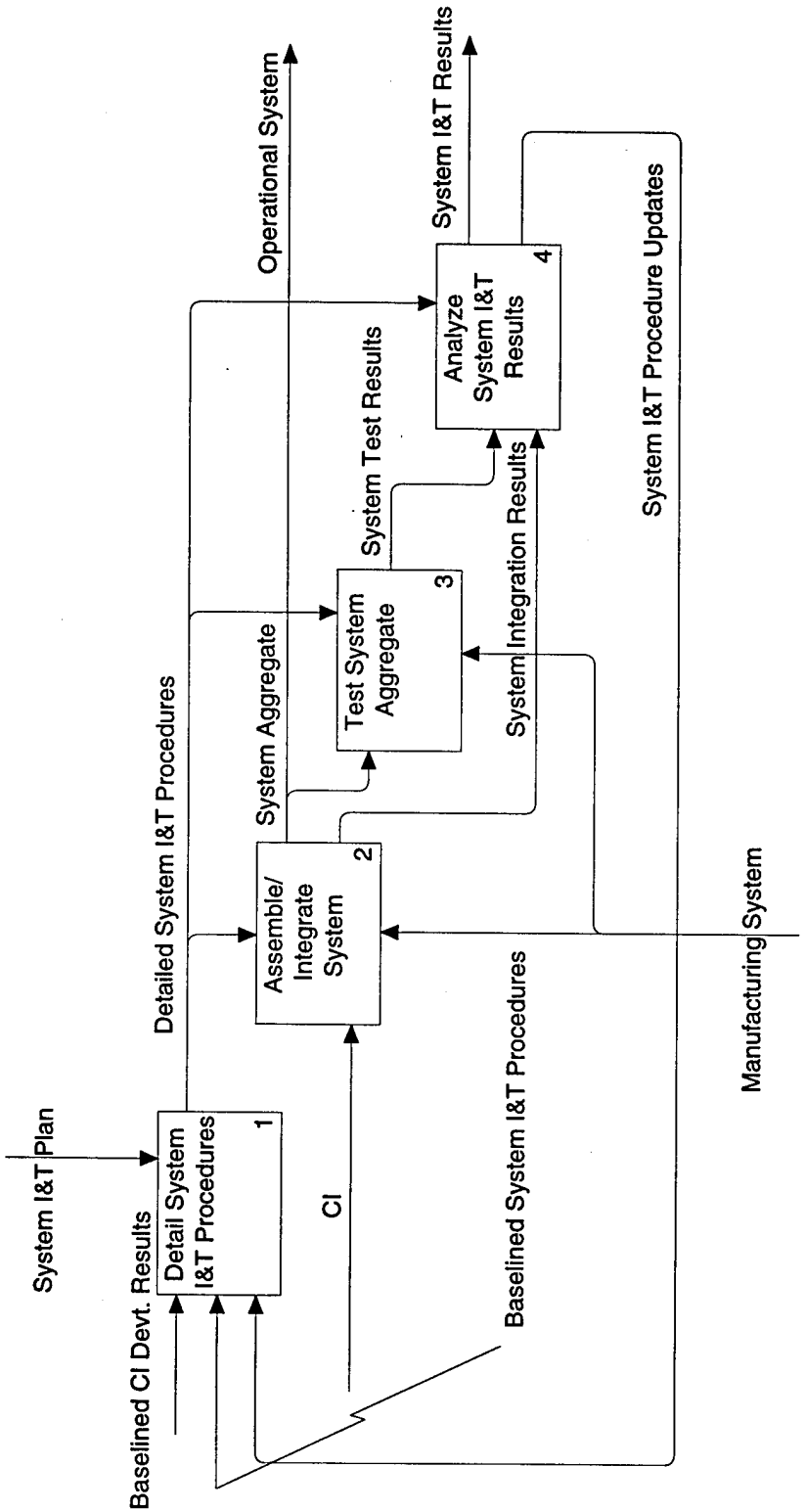


Figure 25. Integrate and Test System (A4)

This page intentionally left blank.

B. ISSEP ACTIVITY DESCRIPTIONS

This appendix contains detailed activity descriptions for the ISSEP model. Section 3.3 contains a high-level description of the activities. The activity descriptions are in alphabetical order.

Analyze CI I&T Results (A344)

Use the CI Integration Results and the CI Test Results to determine whether the Detailed CI I&T Procedures need to be updated. List any failures and their apparent causes and generate the CI I&T Results to be baselined and forwarded to the validation and verification activities of Design and Verify CI.

Analyze CI Requirements (A321)

From the Baselined CI Requirements, ensure that CI requirements are specified with the degree of precision mandated by the selected design method(s). Refine, elaborate, and express the behavioral and performance requirements in a usable form to create the CI Requirements Specification. This specification becomes the basis for what will be developed in this CI. Consider Reusable Assets, if available, when creating the CI Requirements Specification, to leverage existing components as well as requirements specification and design artifacts. Consider and resolve any alternative specification recommendations that may have been initially developed for some requirements, examined in the Evaluate CI Alternatives activity, and documented in the CI Evaluation Results. Analyze and resolve any inconsistencies, omissions, or other errors in the CI Requirements Specification identified in the Validate & Verify CI Solution activity and documented in the CI V&V Results, which is part of the CI Design Results.

Analyze CI Risk (A313)

Identify potential long- and short-term risks, particularly those that affect the current increment. Use the Baselined CI EoS and other parts of the CI Baseline/Plan/Status, as necessary, to determine the significant risks to the development of this increment. The risks identified should be within the scope of the context defined in the EoS. Produce a CI Risk Management Plan, which includes the risks and strategies for risk mitigation.

Analyze System I&T Results (A44)

Use the System Integration Results and the System Test Results to determine whether the Detailed System I&T Procedures need to be updated. List any failures and their apparent causes and generate the System I&T Results to be baselined and forwarded to the validation and verification activities of Design and Verify System.

Analyze System Requirements (A21)

From the Baselined Customer Needs, assess the problems the system is to solve and the needs that the system is to address; define the environment in which the system is to operate; and hence, define the requirements that the system must satisfy to be acceptable to the user and customer of the system. Use the work products contained in the Reusable Assets where they can be adapted to the system under development. Include behavioral requirements that ensure that the system will generate appropriate responses to system inputs and events. Derive performance requirements from the needs and assess them against their effect on the system's ability to meet the customer/user requirements. After the behavioral and performance requirements are established, merge them to define the System Requirements by identifying the behavioral requirements (if any) that the performance requirements constrain. Refine the requirements by analyzing their consistency and ensure that they represent identified customer expectations and project constraints by accommodating the System Design Results.

Analyze System Risk (A13)

Identify potential long- and short-term risks, particularly those that affect the current increment. Use the Baselined EoS and other parts of the System Baseline/Plan/Status, as necessary, to determine which are critical to the increment development effort and when mitigation action is recommended. The risks identified should be within the scope of the context defined in the EoS.

Develop a set of mitigation strategies for each risk and a time table for implementing them. Produce a Risk Management Plan, which includes the risks and strategies for risk mitigation.

Assemble/Integrate CI (A342)

Progressively assemble and integrate the Components according to the Detailed CI I&T Procedures. Provide the CI Aggregate for testing and provide the CI Integration Results for analysis. Continue until the aggregate becomes the complete CI.

Assemble/Integrate System (A42)

Progressively assemble and integrate the CIs according to the Detailed System I&T Procedures. Provide the System Aggregate for testing and provide the System Integration Results for analysis. Continue until the aggregate becomes the complete Operational System.

Control CI Baseline (A311)

Baseline CI Requirements, CI Development Plan, CI Development Results, and all management and technical products created during this increment. Review every product to determine whether it qualifies for baselining and, if accepted, add to the current baseline information in the CI Baseline/Plan/Status, which is output. Track implementation of changes to ensure that product configuration is controlled. Identify and include, in each baseline, all nontangible parts of the developing CI (e.g., design and end-user documentation, software, and all the planning and status information) necessary for the recreation of the baseline. Track all subsequent changes to baselines as part of the configuration status. Maintain the history of changes to each baseline.

The rigor of the review and baseline functions are contained in the CI Development Plan and range from an informal review and creation of a new version of the CI Baseline/Plan/Status to a formal acceptance and formal configuration management. As part of the review activity, note changes to the previous baseline and store, track, and analyze these changes as part of the planning process.

With each increment, the product baseline grows (e.g., the new product and associated plans and documentation are added to the previous baseline). If it is necessary to make changes to any part of a baseline, make those changes in the appropriate activities and update and version the revised products in the Control CI Baseline activity so that this activity always maintains the "latest and greatest" version of all the baselined products.

Control System Baseline (A11)

Baseline Customer Needs and all management and technical products created during the current increment, System Development Plan, and System Development Results to establish a product baseline for the system. Review every product to determine whether it qualifies for baselining and, if accepted, add to the current system baseline, which is output. Track implementation of changes to ensure that product configuration is controlled. Identify and include, in each baseline, all nontangible parts of the developing system (e.g., design and end-user documentation, software, and all the planning and status information) necessary for the recreation of the baseline. Track all subsequent changes to baselines as part of the configuration status. Maintain the history of changes to each baseline.

The rigor of the review and baseline functions are contained in the System Development Plan and range from an informal review and creation of a new version of the system baseline to a formal acceptance and formal configuration management. As part of the review activity, note changes to the previous baseline and store, track, and analyze these changes as part of the planning process.

With each increment, the product baseline grows (e.g., the new product and associated plans and documentation are added to the previous baseline). If it is necessary to make changes to any part of a baseline, make those changes in the appropriate activities and update and version the revised products in the Control Baseline activity so that this activity always maintains the "latest and greatest" version of all of the baselined products.

Define Functional Architecture (A22)

Create a Functional Architecture, made up of a hierarchy of functions and their internal behavior and interfaces, by partitioning the System Requirements. The interfaces can be electrical, mechanical, or logical, and they define the interactions of the functions with each other as well as with the external environment. Use criteria that include performance and design considerations and begin to focus on a solution. Identify alternative feasible solutions that meet the requirements. Use the adaptable functional architectures contained in the Reusable Assets where they can be adapted to the system under development. Update the Functional Architecture, as necessary, to accommodate selections and improvements identified in the System Evaluation Results and to ensure that it is compliant with the System Requirements according to the System V&V Results.

Design and Verify CI (A32)

Develop a validated and verified design for the CI by analyzing the Baselined CI Requirements, and selecting from among alternative architectural and detailed designs to obtain an optimum solution. Use the adaptable CI requirements and architectural and detailed designs contained in the Reusable Assets if appropriate. Document the design, including component requirements, and any risks, in the CI Design & Verification. Generate CI I&T Procedures for later use by the Integrate and Test CI activity. After CI integration and test is complete, verify the Baselined CI I&T Results and determine whether the CI is ready for system integration.

Design and Verify System (A2)

Evolve a System Design from the Baselined Customer Needs by analyzing those needs to define the System Requirements and selecting from alternative functional and physical architectures to obtain an optimum design solution. Use the adaptable system requirements and functional and physical architectures contained in the Reusable Assets if they can be adapted to the system under development. Generate System I&T Procedures for later use by the Integrate and Test System activity. After integration and test is complete, verify the System Test Results and determine whether the system is ready for delivery to the customer.

Detail CI I&T Procedures (A341)

Using implementation details from the Component Baseline, which is part of the CI Baseline/Plan/Status, elaborate on the Baselined CI I&T Procedures. Take into account any CI I&T Procedure Updates fed back by the Analyze CI I&T Results activity. Produce a set of Detailed CI I&T Procedures explaining exactly how the components are to be integrated and tested and how the results are to be analyzed.

Detail System I&T Procedures (A41)

Using implementation details from the CI Designs and the CI I&T Results, which are parts of the Baselined CI Devt. Results in the System Baseline/Plan/Status, elaborate on the Baselined System I&T Procedures. Take into account any System I&T Procedure Updates fed back by the Analyze System I&T Results activity. Produce a set of Detailed System I&T Procedures explaining exactly how the CIs are to be integrated and tested and how the results are to be analyzed.

Develop Component (A33)

Create a successfully tested Component that meets the Component Requirements, which were generated in the Design and Verify CI activity and baselined in the Manage CI Development activity as part of the CI Baseline/Plan/Status. Use the Component Development Plan part of the CI Baseline/Plan/Status to control the development activities and document any status information and risks in the Component Devt. Results/Status. In the case of a hardware component, use the input Parts and Materials to incorporate purchased parts, and use the mechanism Manufacturing System to perform fabrication. Use or modify existing components contained in the Reusable Assets if they can be adapted to the component under development.

Develop Configuration Item (A3)

Create an integrated and successfully tested Configuration Item that meets the CI Requirements generated in the Design and Verify System activity and baselined in the Manage System Development activity as part of the System Baseline/Plan/Status. Use the System Baseline/Plan/Status to control the development activities and document any status information and risks in the CI Baseline/Plan/Status. In the case of a hardware configuration item, use the input Parts and Materials and the mechanism Manufacturing System to create any tangible components. Use the adaptable CI requirements and architectural and detailed designs contained in the Reusable Assets if they can be adapted to the CI under development.

Develop Operational System (A0)

Produce an Operational System that meets the Customer Needs by following the Organization Plan/Status, using Parts and Materials as required, and taking advantage of existing Reusable Assets where appropriate. Also produce the System Baseline/Plan/Status as supporting information to describe the current development effort. Use the Development Environment to support the development process and the Manufacturing System to perform any hardware fabrication or to provide special tools or test equipment.

Develop Unit Test Cases (A332)

Develop the Unit Test Cases for the component and specify the order in which they will be run. Base the Unit Test Cases on both the Component Requirements and the structure of the Component itself, since unit testing is usually "white box." Take into account the prior test results from the Component Devt. Results/Status.

Develop/Update CI Plan (A316)

Use the Baselined CI Increment Results to define the CI Development Plan Update. The CI Development Plan is a long-term plan that defines each of the development increments at a high level, including the planning constraints for the next level of decomposition (e.g., components). If a CI Development Plan already exists, update the plan based on the results of the past increment development efforts, including lessons learned, newly identified risks, and status information. If this is the first increment, generate the first version of this plan from the context information and risk analysis results contained in the CI Baseline/Plan/Status. This plan is a living document and is kept accurate and current.

Develop/Update System Plan (A16)

Use the Baselined Increment Results to define the System Development Plan Update. The System Development Plan, a long-term plan that defines each of the development increments at a high level, including the planning constraints for the next level of decomposition (e.g., CIs). If a System Development Plan already exists, update the plan based on the results of the past increment development efforts, including lessons learned, newly identified risks, and status information. If this is the first increment, generate the first version of this plan from the context information and risk analysis results contained in the System Baseline/Plan/Status. This plan is a living document and is kept accurate and current.

Evaluate CI Alternatives (A324)

Compare alternatives to determine which best meets the Baselined CI Requirements (both functional and performance requirements) within the constraints of the CI Design Plan, and suggest improved alternatives where appropriate. Focus the evaluation on alternative specifications for a requirement (or requirements), alternative architecture (or architectural elements), or alternative designs (or design components), as appropriate. Document the results of the evaluation in the CI Evaluation Results and, when no design risks requiring immediate resolution remain, document the selected alternative as part of the CI Design, which becomes part of the CI Design & Verification.

Evaluate System Alternatives (A24)

Perform trade studies of the alternative functional architectures to select the arrangement that best supports the identified Baselined Customer Needs and System Requirements. Identify improvements that would lead to a better Functional Architecture. Analyze the physical solution alternatives to determine which one best satisfies the allocated functional and performance requirements, interface requirements, and design constraints. Identify improvements that would lead to a better Physical Architecture. Collect the System Evaluation Results to document the studies and proposed improvements. Establish the System Design based on a selected architecture, including preliminary specifications of the CSCIs, HWCIs and interfaces, as well as a record of requirements, alternatives, and design decisions leading to the selection.

Implement Component (A331)

Implement the Component to meet the Component Requirements, as controlled by the Component Development Plan. Use or modify existing components or parts contained in the Reusable Assets if they can be adapted to the component under development. For software components, perform the coding and, if required by the plan, evolve the component over multiple builds of the CI. For hardware components, use the input Parts and Materials to incorporate purchased parts and the mechanism Manufacturing System to perform fabrication and assembly. In either case, incorporate the prior test results from the Component Devt. Results/Status. Include the implementation decisions, status, lessons learned, and any newly identified risks with the implemented Component. The Component is the hardware unit or the executable version of the software unit on appropriate electronic media.

Integrate and Test CI (A34)

Assemble and test the Components according to the Baseline CI I&T Procedures, which include test cases and expected results, and document the outcome in the CI I&T Results, which describe the status of the integration and test process and any exceptions to the observations expected by the procedures. Use the Manufacturing System mechanism to provide any tools and test equipment required. When the integration and testing is successful and no issues remain, the CI is ready for verification in the Design and Verify CI activity.

Integrate and Test System (A4)

Assemble and test the hardware and software Configuration Items according to the Baselined System I&T Procedures, which include test cases and expected results, and document the outcome in the System

I&T Results, which describe the status of the integration and test process and any exceptions to the observations expected by the procedures. Use the Manufacturing System mechanism to provide any tools and test equipment required. When the activity has been successfully completed, the Operational System is ready for verification in the Design and Verify System activity. The Operational System is the tangible part of the system. Nontangible items such as software and user documentation are contained in the System Baseline/Plan/Status.

Manage CI Development (A31)

Plan, control, and coordinate the development of the CI by managing the CI-level activities. Use the System Baseline/Plan/Status, which includes the CI Requirements as a basis for creation of the CI Baseline/Plan/Status, including the definition of the CI development increments. The System Development/Plan/Status provides the scope and context, constraints, and the high-level planning considerations for development of this CI. Gather the status information for this output from the CI Development Results, which include the CI Design & Verification, the CI I&T Results, and the Component Devt. Results/Status. Also use the results of the design, development, and integration and test activities to produce and maintain the baseline information, which grows as CI development progresses to include all nontangible parts of the CI. Repeat the management activities every increment, so that the CI context, risks, and plan are reviewed, and the plan is modified to reflect the current status and updated, as necessary, to ensure that this part of the project remains focused on critical project objectives.

Manage System Development (A1)

Plan, control, and coordinate the development of the system by managing the system-level activities. Use the Organization Plan/Status and the Customer Needs to define the system context (e.g., objectives, goals, stakeholders) and analyze system risks as a basis for creating the System Baseline/Plan/Status, including the definition of the system increments. Gather the status information for this output from the System Development Results, which include the System Design & Verification, the System I&T Results, and the CI Baseline/Plan/Status. Also use the results of the design, development, and integration and test activities to produce and maintain the baseline information, which grows as system development progresses to include all nontangible parts of the system. Repeat the management activities every increment, so that the system context, risks, and plan are reviewed and modified to reflect the current status to ensure that the project remains focused on critical project objectives.

Perform Architectural Design (A322)

Design an architecture that satisfies the CI Requirements Specification, defining a set of components and their interrelationships and allocating requirements to them. The components of the architecture represent pieces of the functionality (or requirements) allocated to the CI. Specify the dependencies, input/output behavior, and performance constraints (e.g., throughput, stimulus response time) of each component. The relationships among the components represent the interfaces between them and any assumptions/constraints placed on these interfaces. Identify alternative feasible solutions that meet the requirements. Use any existing components or architectural fragments from the Reusable Assets where they can be adapted to the CI under development. Refine the architecture based on recommendations from the Evaluate CI Alternatives activity, which are documented in the CI Evaluation Results. Analyze

and resolve any architecture risks identified in the Validate & Verify CI Solution activity and documented in the CI V&V Results, which are part of the CI Design Results.

Perform Detailed Design (A323)

Refine the CI Architecture, specifying the internal structure of the components. Include mandated algorithms, data structures, or code fragments (either existing or to-be-developed), details of internal logic (e.g., conditional paths of execution and the timing allocations for each), and any other constraints on the internal design, resulting in a CI Detailed Design that defines the viable alternatives. Consider adaptation and use of available specifications or specification fragments from the Reusable Assets. Consider and resolve any alternative design recommendations input from the Evaluate CI Alternatives activity, which are documented in the CI Evaluation Results. Analyze and resolve any design issues identified in the Validate & Verify CI Solution activity and documented in the CI V&V Results.

Perform Unit Testing & Analysis (A333)

Exercise the Unit Test Cases and analyze the test results to ensure that implementation of the component is complete and consistent with respect to the Component Requirements. Generate the Component Devt. Results/Status, which includes the implementation decisions and rationale; the test cases; the results of the testing and the associated analysis; newly identified risks; and, if this is a software component, the source code.

Plan CI Increment Development (A314)

Use the Baselined CI RMP, the CI Increment Requirements, and any required additional information contained in the CI Baseline/Plan/Status to determine how best to reach the increment objectives and mitigate risk. First, establish development goals for the increment and use them as a basis for selecting a development strategy. Make detailed size, cost, and schedule estimates. Tailor and instantiate the development process for the increment and document detailed work assignments. Hence, develop the detailed development plan for the next increment. This plan is the CI Increment Plan, a portion of the CI Development Plan, which becomes part of the CI Baseline/Plan/Status. This detailed plan remains within the scope of the CI Development Plan, with added detail to make it enactable.

Plan Sys. Increment Development (A14)

Use the Baselined Risk Management Plan, the Increment Requirements, and any required additional information contained in the System Baseline/Plan/Status to determine how best to reach the increment objectives and mitigate risk. First, establish development goals for the increment and use them as a basis for selecting a development strategy. Make detailed size, cost, and schedule estimates. Tailor and instantiate the development process for the increment and document detailed work assignments. Hence, develop the detailed development plan for the next increment. This plan is the Increment Plan, a portion of the System Development Plan, which is part of the System Baseline/Plan/Status. This detailed plan remains within the scope of the System Development Plan, with added detail to make it enactable.

Synthesize Physical Architecture (A23)

Allocate the System Requirements and the elements of the Functional Architecture to a Physical Architecture that defines the viable alternatives in terms of hardware, software, and people (procedures). Define interfaces that communicate the interactions between the parts of the system, and define technical parameters that drive the performance of the parts. Use the adaptable physical architectures contained in the Reusable Assets where they can be adapted to the system under development. Identify alternative feasible solutions that implement the requirements and functions. Update the Physical Architecture, as necessary, to accommodate selections and improvements identified in the System Evaluation Results, and to ensure that the physical architecture conforms to the Functional Architecture and System Requirements, according to the V&V Results.

Test CI Aggregate (A343)

After each step of integration, as defined by the Detailed CI I&T Procedures, perform the specified tests on the current CI Aggregate, up to and including the CI itself. Use independent personnel to perform any required CI qualification testing. If needed, perform inspection of incoming component-level COTS parts. Provide the CI Test Results for analysis.

Test System Aggregate (A43)

After each step of integration, as defined by the Detailed System I&T Procedures, perform the specified tests on the current System Aggregate, up to and including the Operational System itself. Use independent personnel to perform any required system qualification testing. If needed, perform inspection of incoming CI-level COTS parts. Provide the System Test Results for analysis.

Track CI Increment Development (A315)

Control the enactment of the Baselined CI Increment Plan and ensure that the development progresses accordingly. Use the CI Development Results to assess progress against the plan and analyze and/or resolve development issues. Minor modifications to the plan are permitted, but if major replanning is necessary, or when the development goals documented in the plan are met, terminate this activity and initiate the Develop/Update CI Plan activity. Produce CI Increment Results that contain the actual development measures and any risks that were identified and/or resolved during the increment.

Track Sys. Increment Development (A15)

Control the enactment of the Baselined Increment Plan and ensure that the development progresses accordingly. Use the System Development Results to assess progress against the plan and analyze and/or resolve development issues. Minor modifications to the plan are permitted, but if major replanning is necessary, or when the development goals documented in the plan are met, terminate this activity and initiate the Develop/Update System Plan activity. Produce Increment Results that contain the actual development measures and any risks that were identified and/or resolved during the increment.

Understand CI Context (A312)

Identify factors that that could have an influence on the success of CI development. Define the scope of this increment of the CI development. Analyze the Baselined CI Requirements and other parts of the CI Baseline/Plan/Status, as necessary, to determine the factors that influence the success of this increment of the CI development. Determine the increment objectives and constraints and identify alternatives for meeting the objectives while remaining within the constraints. Identify the stakeholders, for the increment. Produce or update the CI Estimate of the Situation that documents the context for use in managing the increment development.

Understand System Context (A12)

Identify factors that that could have an influence on the success of system development. Define the scope of this increment of the system development. Analyze the Baselined Customer Needs and other parts of the System Baseline/Plan/Status, as necessary, to determine the factors that influence the success of this increment. Determine the increment objectives and constraints and identify alternatives for meeting the objectives while remaining within the constraints. Identify the stakeholders, for the increment. Produce or update the Estimate of the Situation that documents the context for use in managing the increment development.

Validate & Verify CI Solution (A325)

Validate the CI Requirements Specification to ensure that it adequately represents the Baselined CI Requirements and is complete and consistent. Assess the completeness of the CI Architecture in satisfying the validated requirements. Verify that the CI Detailed Design is traceable to the verified CI Architecture as well as to the validated CI Requirements Specification. Generate the CI I&T Procedures, which describe how the components are to be progressively assembled and tested to determine compliance of the integrated CI with the CI Requirements Specification and the CI Design. Analyze the Baselined CI I&T Results to determine whether any changes have to be made to the requirements, architecture, or detailed design. Verify that the CI is complete. Identify any inconsistencies, omissions, ambiguities, or areas for concern, and document them in the CI V&V Results, along with all verification and validation completed on any of the work products produced in the design of the CI, including the results of testing the CI itself.

Validate & Verify System Solution (A25)

Evaluate the System Requirements to ensure that they represent identified Customer Needs and project constraints and that all operations and support concepts have been fully addressed. Assess the completeness of the Functional Architecture in satisfying the validated requirements. Verify that the Physical Architecture is traceable to the verified Functional Architecture as well as to the validated System Requirements. Generate System I&T Procedures, which describe how the hardware and software CIs are to be progressively assembled and tested to determine compliance of the integrated system with the System Requirements and System Design. Analyze the Baselined System I&T Results, which document the outcome of the Integrate and Test System activity, to determine whether any changes have to be made to the requirements or architecture of the system. Verify that the system is complete and ready for delivery. Produce the System V&V Results to document all verification and validation

completed on any of the work products produced in the design of the Operational System, including the results of testing the system itself.

This page intentionally left blank.

C. ISSEP INFORMATION FLOW DESCRIPTIONS

This appendix contains detailed information flow descriptions for the ISSEP model. The information flows are in alphabetical order.

Alternatives & Customer Needs

The Alternatives & Customer Needs provide the work products of the system design activities for the evaluation and verification activities to identify selected alternatives and to report whether they are compliant.

Includes:
Baselined Customer Needs
System Requirements
Functional Architecture
Physical Architecture

Baselined CI Devt. Results

The Baselined CI Devt. Results are part of the System Baseline/Plan/Status and are a version of the CI Development Results included in the CI Baseline/Plan/Status. They include the CI Design & Verification, Component Devt. Results/Status, and the CI I&T Results and provide implementation details for the Detail System I&T Procedures activity.

Included in:
System Baseline/Plan/Status

Baselined CI EoS

The Baselined CI EoS, Estimate of the Situation, defines the mission of the CI and its development, the relationships of the other system units involved in the development of the CI, and relationships with stakeholders. This information flow documents objectives, assumptions, and constraints on the development of the CI. The objectives can be political, technical, organizational, and/or economic. Assumptions include stakeholder expectations, how interactions are to be handled with other system units, and how the development will be staffed.

Included in:
CI Baseline/Plan/Status

Baselined CI I&T Procedures

The Baselined CI I&T Procedures describe how the components that make up the CI design are to be progressively assembled and tested to determine compliance with the CI Requirements. The procedures include the test cases and expected results. These

procedures are the baselined version of the CI I&T Procedures that were generated in the Design and Verify CI activity, bundled as part of the CI Design & Verification and baselined in the Manage CI Development activity as part of the CI Baseline/Plan/Status.

Included in:
CI Baseline/Plan/Status

Baselined CI I&T Results

The Baselined CI I&T Results document the outcome of the Integrate and Test CI activity. This information flow includes issues, concerns, status information, as well as the results from executing the test cases. The CI I&T Results becomes part of the CI Baseline/Plan/Status in the Manage CI Development activity and is used in the Design and Verify CI activity as input to the Validate & Verify CI Solution activity where it is used to perform an analysis of the final integrated CI and determine whether it is complete.

Included in:
CI Baseline/Plan/Status

Baselined CI Increment Plan

The Baselined CI Increment Plan documents the development goals and associated success criteria that support the objectives for the CI that are documented in the CI Baseline/Plan/Status. This information flow defines the estimated size and scope of the development for the current increment; development cost and schedule for each activity planned for the increment; resources allocated to each activity in the increment; methods, tools, and facilities needed to complete the increment's activities; sequence and dependencies between the increment's activities; and the work breakdown structure (WBS) for the activities in the current increment.

Included in:
CI Baseline/Plan/Status

Baselined CI Increment Results

The Baselined CI Increment Results include current plan-to-actual cost information, schedule progress, and risk management information. This information flow also contains information about issues and concerns that have been identified and an analysis of their potential impact.

Included in:
CI Baseline/Plan/Status

Baselined CI Requirements

The requirements for each of the CIs in the system are generated in the Design and Verify System activity. The requirements for each CI, CI Requirements, are baselined in the Manage System Development activity as part of the System Baseline/Plan/Status and are passed down into the appropriate Develop Configuration Item activity (i.e., each Develop Configuration Item activity gets a unique set of CI Requirements). The CI Requirements are then baselined at the CI level in the Manage CI Development

activity, become part of the CI Baseline/Plan/Status, and flow into the Design and Verify CI activity.

Included in:
CI Baseline/Plan/Status
CI Requirements/Alternatives

Baselined CI RMP

The Baselined CI RMP, documents the identified risks, potential risk mitigation strategies, selected risk mitigation strategies and the rationale for their selection, and the implementation plan for the selected strategies.

Included in:
CI Baseline/Plan/Status

Baselined Customer Needs

The Baselined Customer Needs consist of a baselined version of the Customer Needs, which define the customers' and stakeholders' (e.g., users, acquirers, manufacturing, contractor, subcontractors, developers) goals for the system from its conception until it is decommissioned. This information flow also defines the reasons for the system's existence. This flow defines the operational concept that describes how the system is intended to function, the measures of effectiveness of the system, the critical influencing factors, customer requirements, and customer expectations.

Included in:
System Baseline/Plan/Status
Alternatives & Customer Needs

Baselined EoS

The Baselined EoS defines the mission of the system and its development, the relationships of the organizations involved in the development of the system part, and relationships with stakeholders. This information flow documents objectives, assumptions, and constraints on the development of the system part. The objectives can be political, technical, organizational, and/or economic. Assumptions include stakeholder expectations, how interactions are to be handled with other organizational units, and how the development will be staffed.

Included in:
System Baseline/Plan/Status

Baselined Increment Plan

The Baselined Increment Plan documents the development goals and associated success criteria that support the objectives for the system part that are documented in the System Baseline/Plan/Status. This information flow defines the estimated size and scope of the development for the current increment; development cost and schedule for each activity planned for the increment; resources allocated to each activity in the increment; methods, tools, and facilities needed to complete the increment's activities; sequence and dependencies between the increment's activities; and the WBS for the activities in the current increment.

Included in:
System Baseline/Plan/Status

Baselined Increment Results

The Baselined Increment Results include current plan-to-actual cost information, schedule progress, and risk management information. This information flow also contains information about issues and concerns that have been identified and an analysis of their potential impact.

Included in:
System Baseline/Plan/Status

Baselined Risk Management Plan

The Baselined Risk Management Plan documents the identified risks, potential risk aversion/mitigation strategies, selected risk aversion/mitigation strategies and the rationale for their selection, and the implementation plan for the selected strategies.

Included in:
System Baseline/Plan/Status

Baselined System I&T Procedures

The Baselined System I&T Procedures are part of the System Baseline/Plan/Status and are a version of the System I&T Procedures generated by the Design and Verify System activity. This information flow describes how the hardware and software CIs are to be progressively assembled and tested to determine compliance with the System Requirements. Test cases and expected results are included.

Included in:
System Baseline/Plan/Status

Baselined System I&T Results

The Baselined System I&T Results consist of a baselined version of the System I&T Results, which document the outcome of the Integrate and Test System activity; enable the Design and Verify System activity to determine whether any changes have to be made to the requirements or architecture of the system; and verify that the system is complete and ready for delivery.

Included in:
System Baseline/Plan/Status

CI

The CI is the tangible (e.g., hardware with embedded software) integrated and tested Component for this CI. Over time, as more components are developed, the CI will grow and eventually it will include the entire CI. However, it is not necessary to have all the CI components developed before Integration and Test can begin, and the CI created may not initially contain all components.

The nontangible results of the Integration and Test activity, which includes things such as integrated software components, are contained in the CI I&T Results, baselined in the Manage CI Development activity, and become part of the CI Baseline.

Includes:
CI Aggregate

CI Aggregate

The CI Aggregate is the current integrated set of Components, ranging from a single Component to the full CI as integration and testing proceed.

Included in:
CI

CI Architecture

The CI Architecture defines a set of architectural components that are intended to satisfy the requirements defined in the CI Requirements Specification. More that one CI Architecture may be developed for each CI.

Included in:
CI Requirements/Alternatives

CI Baseline/Plan/Status

The CI Baseline/Plan/Status contains all the information about the current CI development effort. This information flow includes planning and associated status information, design information, integration and test information, and software source code which meet the objectives of the CI development. Information from this information flow may be pulled by the Manage CI Development activities as needed.

Included in:
System Development Results

Includes:
Baselined CI Requirements
Baselined CI EoS
Baselined CI RMP
Baselined CI Increment Plan
Baselined CI Increment Results
Baselined CI I&T Procedures
Baselined CI I&T Results
CI Design Plan
CI I&T Plan
CI Increment Requirements
Component Development Plans
Component Requirements
Component Baseline

CI Design

The CI Design contains the validated and verified design (including requirements specification, architecture, and detailed design) for the CI. This information flow contains only the recommended design alternative. Other design alternatives are kept in a design repository for future access.

Included in:
CI Design & Verification

CI Design & Verification

The CI Design & Verification contains the results of the Design and Verify CI activity. This information flow contains the design, procedures for testing the implementation of the design, design decisions, and any issues and concerns associated with the design.

Includes:
CI Design
CI I&T Procedures
CI V&V Results

Included in:
CI Development Results

CI Design Plan

The CI Design Plan is that part of the CI Baseline/Plan/Status which is required to control the Design and Verify CI activities.

Included in:
CI Baseline/Plan/Status

CI Design Results

CI Design Results contains the issues, concerns, and recommendations that resulted from the evaluation of the specification and design alternatives and, if available, the validation and verification of the selected alternative. If more work must be done to create an acceptable alternative, this information is fed back into the activities that created the alternative that was analyzed.

Includes:
CI Evaluation Results
CI V&V Results

CI Detailed Design

The CI Detailed Design defines a set of detailed designs that are intended for each of the CI Architectures defined in the CI Architecture. More that one CI Detailed Design may be developed for each CI Architecture.

Included in:
CI Requirements/Alternatives

CI Development Plan

The CI Development Plan contains the outputs from the management activities collected for baselining in the "plan" portion of the CI Baseline/Plan/Status.

Includes:
CI Estimate of the Situation
CI Risk Management Plan
CI Increment Plan(s)
CI Increment Results
CI Development Plan Update

CI Development Plan Update

If a CI Development Plan already exists, this update incorporates the results of the past increment development efforts, including lessons learned, newly identified risks, and status information. If this is the first increment, this update becomes the first version of the plan, derived from the context information and risk analysis results contained in the CI Baseline/Plan/Status.

Included in:
CI Development Plan

Includes:
CI Design Plan
CI I&T Plan
Component Development Plans

Included in:
CI Baseline/Plan
CI Baseline/Plan/Status

CI Development Results

The CI Development Results are the results of the CI development activities, including the design documents, results of developing the component(s), and the integration and test results. After each technical activity (i.e., Design and Verify CI, Develop Component, and Integrate and Test CI) completes, all the documentation associated with the activity is passed via the CI Development Results flow into the Manage CI Development activity for baselining.

Includes:
CI Design & Verification
Component Devt. Results/Status
CI I&T Results

CI Estimate of the Situation

The CI Estimate of the Situation defines the mission of the CI and its development, the relationships of the other system units involved in the development of the CI, and relationships with stakeholders. This information on flow documents objectives, assumptions, and constraints on the development of the CI. The objectives can be political, technical, organizational, and/or economic. Assumptions include stakeholder

expectations, how interactions are to be handled with other system units, and how the development will be staffed.

Included in:
CI Development Plan

CI Evaluation Results

The CI Evaluation Results contain the issues, concerns, comments, and recommendations that resulted from analyzing the one or more alternatives for each activity in the design process. That is, there will be evaluation results associated with the evaluation of the requirements specification(s), CI architecture(s), and CI detailed design(s).

Included in:
CI Design Results

CI I&T Plan

The CI I&T Plan is that part of the CI Baseline/Plan/Status which is required to control the Integrate and Test CI activities.

Included in:
CI Baseline/Plan/Status

CI I&T Procedure Updates

The CI I&T Procedure Updates are generated by the Analyze CI I&T Results activity to ensure that any changes needed in the Detailed CI I&T Procedures may be incorporated.

CI I&T Procedures

The CI I&T Procedures describe how the components that make up the CI design are to be progressively assembled and tested to determine compliance with the CI Requirements. The procedures include the test cases and expected results.

Included in:
CI Design & Verification

CI I&T Results

The CI I&T Results document the outcome of the Integrate and Test CI activity. This information flow includes issues, concerns, status information, as well as the results from executing the test cases. This flow becomes part of the CI Baseline/Plan/Status in the Manage CI Development activity.

Included in:
CI Development Results

CI Increment Plan

The CI Increment Plan documents the development goals and associated success criteria that support the objectives for the CI that are documented in the CI Development Plan. This information flow defines the estimated size and scope of the

development for the current increment; development cost and schedule for each activity planned for the increment; resources allocated to each activity in the increment; methods, tools, and facilities needed to complete the increment's activities; sequence and dependencies between the increment's activities; and the WBS for the activities in the current increment.

Included in:
CI Development Plan

CI Increment Requirements

The CI Increment Requirements are the technical requirements that must be met by this CI development increment. The Increment Requirements are a subset of the total CI requirements that have been allocated to this CI increment.

Included in:
CI Baseline/Plan/Status

CI Increment Results

The CI Increment Results include current plan-to-actual cost information, schedule progress, and risk management information. This flow also contains information about issues and concerns that have been identified and an analysis of their potential impact.

Included in:
CI Development Plan

CI Integration Results

The CI Integration Results describe the status of the assembly/integration process and any exceptions to the observations expected by the procedures.

CI Requirements

The CI Requirements are the specific technical requirements for each of the CIs to be developed. This flow is generated in the Design and Verify System activity and baselined in the Manage System Development activity as part of the System Baseline/Plan/Status. CI Requirements are also baselined in Manage CI Development and become part of the CI Baseline/Plan/Status.

Included in:
System Baseline/Plan/Status

CI Requirements/Alternatives

The Analyze Requirements, Perform Architecture Design, and Perform Detailed Design activities may produce multiple, acceptable solutions. These solutions are grouped into CI Requirements/Alternatives.

Includes:
Baselined CI Requirements
CI Requirements Specification
CI Architecture
CI Detailed Design

CI Requirements Specification

The CI Requirements Specification contains the technical requirements for the CI being developed. The specification provides the basis for the CI architecture, detailed design, and development.

Included in:
CI Requirements/Alternatives

CI Risk Management Plan

The CI Risk Management Plan documents the identified risks, potential risk mitigation strategies, selected risk mitigation strategies and the rationale for their selection, and the implementation plan for the selected strategies.

Included in:
CI Development Plan

CI Test Results

The CI Test Results describe the status of the testing process and any exceptions to the observations expected by the procedures.

CI V&V Results

The CI V&V Results contain the issues, concerns, comments, and conclusions that resulted from validation and verification of the recommended alternative. There will be V&V results associated with the examination of each of the design activities (i.e., the requirements specification, CI architecture, and CI detailed design).

Included in:
CI Design Results
CI Design & Verification

Component

The Component is the item that was assembled/created during the Develop Component activity. The Component includes both the tangible items (e.g, hardware) that are created from parts and materials produced in the Develop Component activity. Documentation, plans, issues, and other nontangible products created during the Develop Component activity are part of Component Devt. Results/Status.

Component Baseline

The Component Baseline is part of the CI Baseline/Plan/Status and is derived from the Component Devt. Results/Status from each Component of the CI. This information flow includes the Component designs and test results and provides implementation details for the Detail CI I&T Procedures activity.

Included in:
CI Baseline/Plan/Status

Component Development Plan

The Component Development Plan contains all the planning information for work done for a Component. This information flow controls the development of components below the CI level.

Included in:
CI Baseline/Plan/Status

Component Devt. Results/Status

The Component Development Results/Status are the results of the Component development activities, including the test cases and the results of the unit testing and analysis. If implementation of the component is not complete and consistent with respect to the Component Requirements, the Component Devt. Results/Status is fed back to the **Manage CI Development** and/or the **Design and Verify CI** activity for correction. Documentation, plans, issues, and other nontangible products created during the Develop Component activity are part of Component Devt. Results/Status.

Included in:
CI Development Results

Component Requirements

The Component Requirements are the specific technical requirements for each of the components to be developed. This information flow is generated in the Design and Verify CI activity as part of the CI Design and baselined in the Manage CI Development activity as part of the CI Baseline/Plan/Status.

Included in:
CI Baseline/Plan/Status

Customer Needs

The Customer Needs define the customers' and stakeholders' (e.g., users, acquirers, manufacturing, contractor, subcontractors, developers) goals for the system from its conception until it is decommissioned. This information flow also defines the reasons for the system's existence. This flow also defines the operational concept that describes how the system is intended to function, the measures of effectiveness of the system, the critical influencing factors, customer requirements, and customer expectations.

Detailed CI I&T Procedures

The Detailed CI I&T Procedures describe exactly how the Components are to be integrated and tested and how the results are to be analyzed. These procedures elaborate on the Baselined CI I&T Procedures, using implementation details from the Component Baselines and part of the Baselined CI Development Results in the CI Baseline.

Detailed System I&T Procedures

The Detailed System I&T Procedures describe exactly how the CIs are to be integrated and tested and how the results are to be analyzed. These procedures elaborate on the Baselined System I&T Procedures, using implementation details from the CI Designs

and the CI I&T Results and parts of the Baselined CI Development Results in the System Baseline.

Development Environment

The Development Environment defines the tools, methods, and people that will execute the development process.

Estimate of the Situation

The Estimate of the Situation defines the mission of the system and its development, the relationships of the organizations involved in the development of the system part, and relationships with stakeholders. This information flow documents objectives, assumptions, and constraints on the development of the system part. The objectives can be political, technical, organizational, and/or economic. Assumptions include stakeholder expectations, how interactions are to be handled with other organizational units, and how the development will be staffed.

Included in:
System Development Plan

Functional Architecture

The Functional Architecture defines the hierarchy of functions that satisfy the System Requirements. Each function documents the inputs to, outputs from, and internal behavior of the function. There are often performance requirements that are allocated to constrain the behavior of the function. Functional interfaces define the information that flows between functions. These interfaces can be electrical, mechanical, or logical. Interfaces define the interactions of the functions with each other as well as with the external environment. The functions can be identified using techniques such as object-oriented or structural decomposition.

Included in:
Alternatives & Customer Needs

Increment Plan

The Increment Plan documents the development goals and associated success criteria that support the objectives for the system part that are documented in the System Baseline/Plan/Status. This information flow defines the estimated size and scope of the development for the current increment; development cost and schedule for each activity planned for the increment; resources allocated to each activity in the increment; methods, tools, and facilities needed to complete the increment's activities; sequence and dependencies between the increment's activities; and the WBS for the activities in the current increment.

If the objective of this increment is to develop the System Baseline/Plan/Status, then the Increment Plan contains the estimated size, cost, and schedule for development of the plan; the WBS for work done to create the plan, if applicable; the resources allocated to complete the plan; the methods, tools, and facilities needed to complete the plan; and sequence dependencies between the plan development activities.

Included in:
System Development Plan

Increment Requirements

The Increment Requirements are the technical requirements that must be met by this system development increment. The Increment Requirements are a subset of the total system requirements that have been allocated to this system increment.

Included in:
System Baseline/Plan/Status

Increment Results

The Increment Results include current plan-to-actual cost information, schedule progress, and risk management information. This flow also contains information about issues and concerns that have been identified and an analysis of their potential impact.

Included in:
System Development Plan

Manufacturing System

The Manufacturing System is the mechanism by which any hardware fabrication required during the activity Develop Component is accomplished and by which the different tools and test equipment typically required for the Integrate and Test activities are provided.

Operational System

The Operational System is the tangible part of the system and contains the integrated and tested hardware and software that meet the objectives of the entire program as expressed in the Customer Needs.

Includes:
System Aggregate

Organization Plan/Status

The Organization Plan/Status contains all the planning information and the associated status information used to guide and constrain the system development, such as organization structure and objectives, cost and resource constraints, and organizational policies and procedures. This information flow also includes all relevant issues, concerns, and objectives for this system. This plan and its associated status provide context for managing the development of the system.

In summary, the Organization Plan/Status may include an Estimate of the Situation; a Risk Management Plan; key deliverable descriptions; estimates of size, cost, and schedule; reusable entity definitions; status information, issues, and concerns; and lessons learned.

Parts and Materials

Parts and Materials are hardware items (i.e., tangible items) that are used in creating hardware components and parts used in the deployment of software items (e.g., tapes, diskettes, and computer hardware used for development transferred to the target site).

Physical Architecture

The Physical Architecture defines mappings from the Functional Architecture to subsystems, components, people, hardware, and software. This information flow defines where the functions are accomplished and how the interfaces between the people, hardware, and software support the interfaces between the formal system functions. This work product includes the physical layout and potential failure mechanisms.

Included in:
Alternatives & Customer Needs

Reusable Assets

The Reusable Assets consist of adaptable system and CI requirements, functional and physical architectures, CI designs, and developed system parts, which are available to support the development of the operational system. Included are their associated documentation such as requirements, design and design rationale, integration and test plans, unit test cases, results of testing, and management plans and status information. These reusable assets are available for inclusion in the developing system, as needed.

Risk Management Plan

The Risk Management Plan documents the identified risks, potential risk mitigation strategies, selected risk mitigation strategies and the rationale for their selection, and the implementation plan for the selected strategies.

Included in:
System Development Plan

System Aggregate

The System Aggregate is the current integrated set of software and hardware CIs, ranging from a single CI to the full Operational System as integration and testing proceed.

Included in:
Operational System

System Baseline/Plan/Status

The System Baseline/Plan/Status contains all the information about the current system development effort. This information flow includes planning and associated status information, design information, integration and test information, and software source code, which meet the objectives of the entire project. Information from this flow may be pulled by the Manage System Development activities, as needed.

Includes:
Baselined Customer Needs

Baselined EoS
Baselined Risk Management Plan
Baselined Increment Plan
Baselined Increment Results
Baselined System I&T Procedures
Baselined CI Devt. Results
Baselined System I&T Results
System Design Plan
CI Requirements
System I&T Plan
Increment Requirements
System Development Plan/Status

System Design

The System Design identifies the preferred alternative Physical Architecture that is selected after the trade-off analysis. This solution is selected based on a comparison of all alternatives. The System Design includes a record of requirements, alternatives, and design decisions and is used to further engineer or implement the system.

Included in:
System Design & Verification

System Design & Verification

The System Design & Verification includes a record of requirements, alternatives, and design decisions, describes how the hardware and software CIs are to be assembled and tested, and documents the results of the verification and validation completed on the work products of the design activities.

Includes:
System Design
System I&T Procedures
System V&V Results

Included in:
System Development Results

System Design Plan

The System Design Plan is required to control the Design and Verify System activities.

Included in:
System Baseline/Plan/Status

System Design Results

The System Design Results provide feedback to the system design activities from the evaluation and verification activities to identify selected alternatives and to report whether the work products are compliant.

Includes:
System Evaluation Results
System V&V Results

System Development Plan

The outputs from the management activities collected for baselining in the "plan" portion of the System Baseline/Plan/Status.

Includes:
Estimate of the Situation
Risk Management Plan
Increment Plan(s)
Increment Results
System Development Plan Update

System Development Plan/Status

The System Development Plan/Status contains all the planning information and the associated status information for work done to develop the system. This information flow also includes all relevant issues, concerns, and lessons learned from development of each increment of the system. This plan and its associated status provide the information for managing the development of CIs below the system level.

Included in:
System Baseline/Plan/Status

System Development Plan Update

An update of the System Development Plan based on the results of the past increment development efforts, including lessons learned, newly identified risks, and status information. If this is the first increment, this update becomes the first version of the plan based on the context information and risk analysis results contained in the System Baseline/Plan/Status.

Included in:
System Development Plan

System Development Results

The System Development Results contain the results from the development activities: Design and Verify System, Integrate and Test System, and Develop Configuration Item. If the system is composed of multiple levels of subsystem, then development results from subsystems that cannot be resolved at lower levels are included in the System Development Results for this system part.

Includes:
System Design & Verification
CI Baseline/Plan/Status
System I&T Results

System Evaluation Results

The System Evaluation Results include the models; test results, including sensitivity analysis; and assessed risks identified in the analysis of each of the system alternative designs considered. These evaluation results define estimates of system performance.

Included in:
System Design Results

System I&T Plan

The System I&T Plan is that part of the System Baseline/Plan/Status which is required to control the Integrate and Test System activities.

Included in:
System Baseline/Plan/Status

System I&T Procedure Updates

The System I&T Procedure Updates are generated by the Analyze System I&T Results activity to ensure that any changes needed in the Detailed System I&T Procedures may be incorporated.

System I&T Procedures

The System I&T Procedures describe how the hardware and software CIs are to be progressively assembled and tested to determine compliance with the System Requirements. Included are test cases and expected results.

Included in:
System Design & Verification

System I&T Results

The System I&T Results document the outcome of the Integrate and Test System activity, enable the Design and Verify System activity to determine whether any changes have to be made to the requirements or architecture of the system, and verify that the system is complete and ready for delivery.

Included in:
System Development Results

System Integration Results

The System Integration Results describe the status of the assembly/integration process and any exceptions to the observations expected by the procedures.

System Requirements

The System Requirements define the behavioral and performance requirements for the system that, when met, satisfy the system developer's obligations in the production of the system.

Included in:
Alternatives & Customer Needs

System Test Results

The System Test Results describe the status of the testing process and any exceptions to the observations expected by the procedures.

System V&V Results

The System V&V Results document the results of any form of verification and/or validation completed on any work products produced in the design of the operational system, including testing the system itself.

Included in:
System Design Results
System Design & Verification

Unit Test Cases

Unit Test Cases contain the set of test cases necessary to test the Component and the sequence in which the test cases should be applied. In many situations, performing one test case will depend on successful completion of another test case.

D. DEVELOP OPERATIONAL SYSTEM CONTEXT (A-1)

This appendix contains a more complete context diagram containing the **Develop Operational System** activity. (In IDEF0 notation, this diagram is numbered A-1 and becomes the parent diagram for the node A0). The A-1 diagram (see Figure 26) and the discussion that follows provide better understanding of external processes that produce inputs, controls, and mechanisms used by the ISSEP model.

Diagram A-1 is an example context. The activities in the diagram represent a possible and reasonable context for the ISSEP model, but some assumptions have been made regarding the organizational and development factors that are present in this example. A different set of assumptions would generate a different ISSEP context. The discussion in Section D.1 includes a list of the major underlying assumptions and a rationale for why these assumptions were made.

D.1 CONTEXT ASSUMPTIONS

There are three major assumptions that are being made in the A-1 diagram:

- ***Product Lines.*** The operational system defined by the ISSEP model is part of a product line and the product line development includes a **Develop Domain System** activity.
- ***Process Focus.*** The organization that is implementing the ISSEP model for the operational system development has an organizational process focus, and process definition and improvement take place at the highest level of the product-line development (i.e., at the A-1 level).
- ***Additional Systems.*** The manufacturing system (e.g., manufacturing of tooling and test equipment as well as system components) and support system (e.g., support for deploying, maintaining, and disposing of the system) are necessary and sufficient for development of the operational system.

D.1.1 PRODUCT LINES

A product line is a collection of similar existing and potential systems that address a designated business area market. The product-line approach to system development benefits organizations that produce systems that solve a set of similar problems with corresponding similar solutions, as opposed to the conventional approach that emphasizes one-of-a-kind handcrafting of each system. Effective production of product lines requires creating a domain that contains the members of the product line and the associated capability for production of future members (Software Productivity Consortium 1993a, 1995b).

Not all systems are appropriate for inclusion in a product line, even if the organization is structured to take a product-line approach. However, the product-line approach is interesting and becoming a more popular development strategy for organizations attempting to leverage resources in order to gain a market advantage. Section D.3 describes how the A-1 process would be impacted if the product-line approach were not used.

D.1.2 PROCESS FOCUS

Process focus involves establishing a set of organizational activities that are responsible for defining and improving the organization's processes (Paulk et al. 1993). An organizational process focus is an important aspect of improving process maturity. Although process focus does not have to be at the A-1 level, this level is traditionally where processes are defined and organizational improvements are made within the product-line approach. Section D.3 describes how the A-1 process would be impacted if the process focus were not at this level.

D.1.3 ADDITIONAL SYSTEMS

Frequently, processes other than the **Develop Operational System** defined in the ISSEP model are needed when producing large systems. The A-1 diagram includes the processes for the manufacturing and support systems. The manufacturing system process provides manufactured parts needed in the development of the operational system. The support system process provides support for the development of the operational system and continued support for the system after delivery to the customer (e.g., maintenance). These are not the only processes that could be modeled. Other processes such as the proposal process could have been included.

The A-1 diagram includes the manufacturing system process to illustrate where the mechanisms for manufacturing the hardware parts and any test equipment are created when developing an operational system that includes hardware components. The A-1 diagram includes the support system process to illustrate where other nonhardware mechanisms (e.g., technology transfer mechanisms) are created for use in the ISSEP model. Section D.3 describes how the A-1 process would be impacted if the manufacturing and support system processes were omitted.

D.2 DEVELOP PRODUCT LINE (A-1)

Figure 26 defines the **Develop Product Line** activities. The **Develop Operational System,** which is the focus of the ISSEP model, receives inputs, controls, and mechanisms from the other activities either directly or indirectly. All the information flows on the **Develop Operational System** context, A-0, are identified in the A-1 diagram.

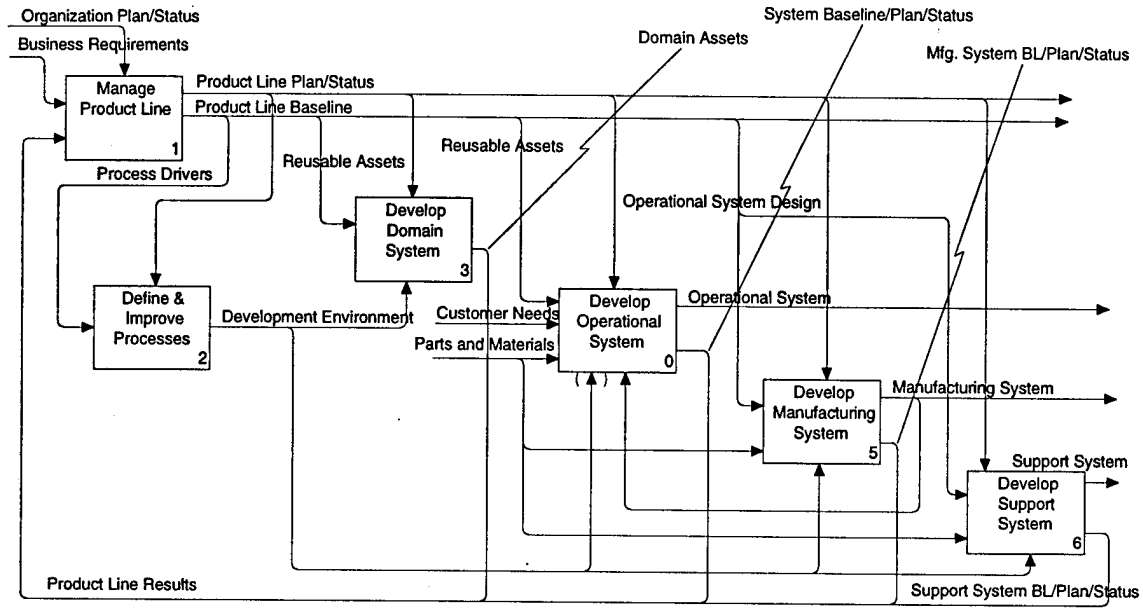


Figure 26. Develop Product Line (A-1)

As with all the other ISSEP model decomposition levels, the **Develop Product Line** activities include a set of management activities, **Manage Product Line**. As with the other ISSEP model management activities, these management activities develop the plans and control the other **Develop Product Line** activities. The **Develop Domain System** activity creates the *Domain Assets*. The *Domain Assets* are system parts that have been designed for reuse and created for use when developing members of the product line (produced in the **Develop Operational System** activity). These assets and the results from the **Develop Operational System**, **Develop Manufacturing System**, **Develop Support System**, **Manage Product Line**, and **Define & Improve Processes** activities are baselined and become *Reusable Assets*. The *Reusable Assets* that were not developed in the **Develop Domain System** activity are available for reuse; but because they were not developed with reuse in mind, they may have limited reuse opportunities.

The following list describes the **Develop Product Line** activities:

- Box 1, **Manage Product Line,** plans, controls, and coordinates the development of the product line. This activity uses the *Organization Plan/Status* and the *Business Requirements* as a basis for planning the product-line development. Status information is gathered from the *Product Line Results.* The updated product line plans, including the status information, are bundled to create the *Product Line Plan/Status.* The *Reusable Assets,* which are part of the *Product Line Baseline,* grow as the product line expands with the addition of newly developed assets. The *Product Line Baseline* also contains the *Operational System Design* and all the other designs, plans, status, test, and related documentation created by the operational, manufacturing, and support systems.

- Box 2, **Define & Improve Processes** defines the *Development Environment* used for creation of the operational, manufacturing, and support systems. The development environment consists of processes, methods, tools, and facilities. This activity is often included at the product-line level so that a standard environment is developed, which can be applied across the entire product line.

Some of the key activities that are considered part of the **Define & Improve Processes** activity include the assessment of the organizational processes, benchmarking of best practices, defining and modifying the standard processes, selecting standard tools and methods, and developing and providing training to ensure the necessary skill levels. The input *Process Drivers* include unique tailoring requirements for each system development and lessons learned and related process metrics that enable the development environment to be continuously improved. The development environment is baselined and controlled in a similar way to the actual systems being developed to ensure the integrity of the environment.

- Box 3, **Develop Domain System**, creates *Domain Assets* such as hardware, software, procedures, tools, and facilities, which can be used in developing members of the product line. The purpose of this activity is to provide a source of easily adaptable, reusable parts. This is accomplished through an analysis of the domain followed by the creation of parts that were identified as providing the most promising opportunities for reuse.

- Box 0, **Develop Operational System**, creates the operational system, including both the design and build of the operational system, and outputs the *Operational System* and the *System Baseline/Plan/Status.* The control into this activity is the *Product Line Plan/Status,* which directs the development of the operational system, and one input is the *Reusable Assets,* which may be used in the system development, if appropriate. Note that in the A-0 context, the control into the **Develop Operational System** activity is the *Organization Plan/Status*, which on this diagram is the control into the **Manage Product Line** activity. This difference is the result of how the product-line activities, and in particular how the management of the product-line activities, have been modeled in this A-1 context. If there were no product-line activities in this A-1 context, then the *Organization Plan/Status* would directly control the **Develop Operational System**, and the A-0 and A-1 contexts would have identical controls.

- Box 5, **Develop Manufacturing System,** creates elements such as the manufacturing tooling, test equipment, processes, procedures, and facilities used in the development of the operational system. The outputs of this activity include the manufacturing system design baseline and associated plans and status, *Mfg. System BL/Plan/Status,* and its physical realization, *Manufacturing System.* The *Manufacturing System* is a mechanism to implement the build process within the **Develop Operational System** activity. This activity includes the inputs *Operational System Design* and *Parts and Materials.* In order to ensure that the operational system can be built within design-to-cost goals, effective communication channels must be maintained between the **Develop Operational System** and **Develop Manufacturing System.**

- Box 6, **Develop Support System**, produces the support system for deploying, maintaining, and disposing of the operational system. Support systems are needed to implement these activities. The output *Support System BL/Plan/Design* includes both the support system design baseline and associated plans and status and the physical realization of the support systems, *Support System.*

D.3 CHANGING ASSUMPTIONS

The following descriptions provide insight on how the process defined by the A-1 diagram would be impacted if a product-line approach is not used.

Product Lines. If the operational system defined by the ISSEP model is not part of a product-line development effort, the **Develop Domain System** activity box in Figure 26 would be omitted. Because this activity develops the *Domain Assets,* which is the major contributor of *Reusable Assets*, the opportunities for reuse in the development of the operational system would be greatly reduced. Without the *Domain Assets*, the only *Reusable Assets* are those created by the **Develop Operational System** activity. Other mechanisms for reuse, such as a reuse repository, could be added to make opportunistic reuse of these assets possible, or *Reusable Assets* could be removed from the ISSEP model.

Process Focus. If the process focus implemented by the **Define & Improve Processes** activity is omitted from the A-1 diagram, the *Development Environment* it creates must be produced by another activity. Organizations that are not producing product lines may choose to model the creation of the *Development Environment* as part of creation of the support system. If the *Development Environment* is created by the support system, it would still be modeled in Diagram A-1 as a mechanism into the **Develop Operational System** activity.

Additional Systems. If the systems created by the manufacturing and support systems are not required for the development of the operational system, they can be omitted from the A-1 diagram without any significant impact. However, if manufacturing or support is required, either these activities need to remain part of the A-1 diagram, or other activities that can generate the requirements must be substituted.

Many of the assumptions stated in Section D.1 are appropriate when developing large, complex systems. As stated in Sections 5.1 through 5.3, process drivers impact how the process is tailored. Assumptions like those stated in D.1 are important process drivers, and if they change, the A-1 context must be modified accordingly. However, this description of the A-1 context should provide a high-level understanding of where and how the major inputs, controls, and mechanisms for the ISSEP model are generated.

This page intentionally left blank.

E. TOOL SUPPORT ENVIRONMENT

This appendix describes the tool support available for the ISSEP model. The appendix begins with a brief description of the tool selection process that was used to choose a tool for developing the ISSEP model. Section E.2 defines the ISSEP model's electronic format, and Section E.3 suggests tool support options for the ISSEP model.

E.1 TOOL SELECTION

Before the requirements for a tool could be formulated, a model representation notation had to be selected. The following notations were considered:

- Data flow diagrams, widely used and understood by software engineers
- Functional flow block diagrams, widely used and understood by systems engineers
- PERT charts, widely used and understood by managers
- IDEF0 diagrams, widely used and understood by process developers

Each notation has its interested parties among the community addressed by ISSEP, but the choice of IDEF0 was largely based on its wide acceptance for use in developing structured graphical representations of a system.

The tool requirements then became the following:

- Support IDEF0 graphics, dictionaries, and rule checking
- Run under Windows
- Permit cut-and-paste to Microsoft Word (Word)
- Be low cost and user friendly

A dedicated IDEF0 tool was preferred over the combined use of a word processor, a drawing tool, and a database tool. Popkin's SA/BPR tool (Popkin Software and Systems, Incorporated 1991–1995) was selected. This tool facilitates the generation of a hierarchical set of IDEF0 diagrams, along with the supporting function, data, and object descriptions. SA/BPR provides balancing between parent and child diagrams and generates Interface Description Language (IDL) files that can be used to export IDEF0 models to other IDEF0 tools.

E.2 AVAILABLE ISSEP FORMATS

The ISSEP model is available electronically. The activity and information flow descriptions are available in either Word, Rich Text Format (RTF), or as encyclopedias in the SA/BPR tool. The IDEF0 models are available in IDL, in SA/BPR-compatible formats, or as pict files in Microsoft Office compatible formats (e.g., in Word or PowerPoint compatible files). This report is available in Word or RTF format. The report and model can be ordered from the Software Productivity Consortium Clearinghouse at 1-800-827-4772 or e-mail **brewer@software.org**.

E.3 TOOL SUPPORT FOR ISSEP

Once the ISSEP model has been successfully installed at a user's site, work can begin on tailoring it for the organization. The first step is to understand the process needs of the organization and to use this information to further specify the process and, thus, provide an additional level of detail to the model. Part of this specification includes the incorporation of existing processes and procedures and the identification of necessary automated support for the ISSEP activities. The tools need not be integrated; they can be standalone project management tools, word processors, spreadsheets, requirements traceability tools, design tools, compilers, and debuggers. Alternatively, an entire environment such as I-CASE can be used. ISSEP is not tool dependent, but adding tool support will make implementing many of the activities more practical and less tedious.

This report does not specify ISSEP tool requirements, but a report survey can be found in *A Tailorable Process for Systems Engineering* (Software Productivity Consortium 1995b). Defining the tooling requirements is one of the next steps in further specifying the ISSEP model.

F. MAPPING TO STANDARDS

F.1 MAPPING TO MIL-STD-498

This appendix contains a mapping of the ISSEP model to the detailed requirements in Section 5 of MIL-STD-498 (see Table 6). The nineteen activities contained in MIL-STD-498 are listed in the first column. The nineteenth activity, Other Activities, has been expanded and each Other Activity is a separate row in the matrix. Under the heading ISSEP Activities, the ISSEP model activities are listed and categorized as follows:

- ***ISSEP Management Activities (Management).*** These columns apply to both the system and CI management activities.
- ***Design and Verify System Activities (System)***
- ***Design and Verify CI Activities (CI)***
- ***Develop Component Activities (Comp.)***
- ***Integrate and Test CI Activities (CI I&T)***
- ***Integrate and Test System Activities (Sys. I&T)***

This matrix contains only explicit mappings where a significant part of the ISSEP model activity is performing tasks directly related to the tasks required by the MIL-STD-498 standard. An X in the matrix indicates that the MIL-STD-498 activity in that row is performed in the ISSEP model activity in the corresponding column. Frequently, there are several Xs in a row indicating that there are several ISSEP activities that perform the one MIL-STD-498 activity. Every ISSEP activity is mapped to at least one MIL-STD-498 activity.

However, not every MIL-STD-498 activity is mapped to an ISSEP model activity because, in some cases, there is no direct mapping possible; rather, the mapping is more implicit. For example, the MIL-STD-498 activity Preparing for Software Use could be mapped to all of the following ISSEP model activities:

- Planning activities, **Plan Increment Development** and **Develop/Update Plan**, because preparing for software use must be planned
- **Control Baseline** activity because the manual baselines must be controlled and incorporated into the delivered baseline

- Design activities, **Analyze CI Requirements, Evaluate CI Alternatives,** and **Validate & Verify CI Solution**, because preparing for use is a design consideration

- Integration and test activities, **Detailed CI I&T Procedures, Test Aggregate CI,** and **Analyze CI I&T Results**, because the manuals must be tested to ensure that they accurately reflect the delivered system and software products

However, putting an X in each of these columns would add little guidance because preparing the software for use is only a small part of the tasks performed by these activities. Although some MIL-STD-498 activities, like the example above, can only be mapped implicitly to the ISSEP model, all of the MIL-STD-498 activities have either an implicit or explicit mapping to the ISSEP model.

Table 6. ISSEP/MIL-STD-498 Compliance Matrix

	ISSEP ACTIVITIES																										
	Management						System					CI					Comp.			CI I&T				Sys. I&T			
MIL-STD 498 Requirement	Control Baseline	Understand Context	Analyze Risk	Plan Increment Development	Track Increment Development	Develop/Update Plan	Analyze System Requirements	Define Functional Architecture	Synthesize Allocated Architecture	Evaluate System Alternatives	Validate & Verify System Solution	Analyze CI Requirements	Perform Architectural Design	Perform Detailed Design	Evaluate CI Alternatives	Validate & Verify CI Solution	Implement Component	Develop Unit Test Cases	Perform Unit Testing & Analysis	Detailed CI I&T Procedures	Assemble/Integrate CI	Test CI Aggregate	Analyze CI I&T Results	Detailed System I&T Procedures	Assemble/Integrate System	Test System Aggregate	Analyze System I&T Results
Project planning and oversight		X	X	X	X	X																					
Establishing a software development environment																											
System requirements analysis							X			X	X	X															
System design								X	X	X	X																
Software requirements analysis												X			X	X											
Software design													X	X	X	X											
Software implementation/unit testing																	X	X	X								
Unit integration and testing																				X	X	X	X				
CSCI qualification testing																X						X	X				
CSCI/HWCI integration and testing																								X	X	X	X
System qualification testing											X															X	X
Preparing for software use																											
Preparing for software transition																											
Software configuration management	X																										
Software product evaluation																											
Software quality assurance (SQA)																											
Corrective action																											
Joint reviews					X																						

Table 6, continued

	ISSEP ACTIVITIES: Management						System					CI					Comp.			CI I&T				Sys. I&T			
Other Activities	Control Baseline	Understand Context	Analyze Risk	Plan Increment Development	Track Increment Development	Develop/Update Plan	Analyze System Requirements	Define Functional Architecture	Synthesize Allocated Architecture	Evaluate System Alternatives	Validate & Verify System Solution	Analyze CI Requirements	Perform Architectural Design	Perform Detailed Design	Evaluate CI Alternatives	Validate & Verify CI Solution	Implement Component	Develop Unit Test Cases	Perform Unit Testing & Analysis	Detailed CI I&T Procedures	Assemble/Integrate CI	Test CI Aggregate	Analyze CI I&T Results	Detailed System I&T Procedures	Assemble/Integrate System	Test System Aggregate	Analyze System I&T Results
Risk management			X																								
Software management indicators				X	X	X																					
Security and privacy																											
Subcontractor management	X	X	X	X	X	X																X				X	
Interface with software IV&V agents					X																						
Coordination with associate																											
Improvement of project processes																											

F.2 MAPPING TO THE CAPABILITY MATURITY MODEL

The section contains a mapping of the ISSEP model to the CMM (see Table 7). The Key Process Areas (KPAs) for Levels 2 and 3 in the CMM are listed in the first column. Under the heading ISSEP Activities, the ISSEP model activities are listed and categorized as follows:

- ***ISSEP Management Activities (Management).*** These columns apply to both the system and configuration item management activities.
- ***Design and Verify System Activities (System)***
- ***Design and Verify CI Activities (CI)***
- ***Develop Component Activities (Comp.)***
- ***Integrate and Test CI Activities (CI&T)***
- ***Integrate and Test System Activities (Sys. I&T)***

An X in the matrix indicates that the CMM KPA in that row is performed in the ISSEP model activity in the corresponding column. Frequently, there are several Xs in a row indicating that there are several ISSEP activities that perform one CMM KPA. Every ISSEP activity is mapped to at least one CMM KPA. However, not every CMM KPA is mapped to an ISSEP model activity because, in some cases, there is no ISSEP activity that adequately covers the total requirements of the CMM KPA.

The Level 4 and 5 KPAs have not been included in the matrix because the ISSEP model does not contain any mappings to them.

Table 7. ISSEP/CMM Compliance Matrix

	ISSEP ACTIVITIES																														
	Management							**System**						**CI**						**Comp.**			**CI I&T**					**Sys. I&T**				
	Control Baseline	Understand Context	Analyze Risk	Plan Increment Development	Track Increment Development	Develop/Update Plan		Analyze System Requirements	Define Functional Architecture	Synthesize Allocated Architecture	Evaluate System Alternatives	Validate & Verify System Solution		Analyze CI Requirements	Perform Architectural Design	Perform Detailed Design	Evaluate CI Alternatives	Validate & Verify CI Solution		Implement Component	Develop Unit Test Cases	Perform Unit Testing & Analysis		Detailed CI I&T Procedures	Assemble/Integrate CI	Test CI Aggregate	Analyze CI I&T Results		Detailed System I&T Procedures	Assemble/Integrate System	Test System Aggregate	Analyze System I&T Results	
Capability Maturity Model for Software																															
The Key Process Areas for Level 2:																															
Requirements Management	X					X		X				X		X				X													
Software Project Planning		X	X	X		X																									
Software Project Tracking and					X	X																									
Software Subcontract Management		X	X	X	X	X																		X				X			
Software Quality Assurance				X		X												X													
Software Configuration Management	X			X		X																									
The Key Process Areas for Level 3:																															
Organization Process Focus																															
Organization Process Definition																															
Training Program																															
Integrated Software Management			X	X	X	X																									
Software Product Engineering				X	X	X								X	X	X	X	X		X	X	X		X	X	X	X				
Intergroup Coordination				X	X	X		X						X																	
Peer Reviews				X		X						X						X				X									

LIST OF ABBREVIATIONS AND ACRONYMS

CI	configuration item
CMM	Capability Maturity Model
COTS	commercial off-the-shelf
CPU	central processing unit
CSCI	computer software configuration item
C^3I	command, control, communications, and intelligence
Devt.	development
EIA	Electronic Industries Association
EoS	Estimate of the Situation
ESIS	Engineering Software-Intensive Systems
GSEP	Generic Systems Engineering Process
HW	hardware
HWCI	hardware configuration item
I&T	integration and test
I-CASE	integrated computer-aided software engineering
IDEF0	Integrated Computer-Aided Manufacturing Definition
IDL	Interface Description Language
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers, Inc.
ISO	International Standards Organization
ISSEP	Integrated Systems and Software Development Process

KPA	Key Process Area
NCOSE	National Council on Systems Engineering
PERT	Program Evaluation and Review Technique
SA/BPR	System Architect Business Process Reengineering
SE-CMM	Systems Engineering Capability Maturity Model
RMP	Risk Management Plan
RTF	Rich Text Format
SQA	software quality assurance
SW	software
Sys.	system
V&V	validation and verification
WBS	work breakdown structure

GLOSSARY

Baseline	(1) An approved version of a system, configuration item, or component regardless of media, fixed at a specific time during the development life cycle. (2) To place the collective products of the development activities under version control.
Component	Either of: (1) A software unit which is an element of a software configuration item. (2) An element of the physical or system architecture, specification tree, and system breakdown structure that is a subordinate element to an assembly and may be composed of two or more subcomponents, or parts; or one or more subassemblies and their associated life-cycle processes. In a noncomplex system, the component may be the lowest element (IEEE P1220).
Configuration item	An aggregation of hardware, software, or both that satisfies an end use function and is designated for separate configuration management by the acquirer (MIL-STD-498).
Increment	A subset of development work which, when completed, creates a portion of the system and progresses the effort toward delivery. Increments specify the activities that are to be performed for each of the developing system parts.
Nontangible item	Developed items such as software, design, and user documentation that are the baselined portions of a delivered system.
Process interface	Defines the information flows between software- and system-centered activities.
Process tailoring	Creating a specific process from a general one.

Product interface	Defines the points within the developing system where the system and software parts interact.
Risk	A measure of the uncertainty of attaining a goal, objective, or requirement pertaining to technical performance, cost, and schedule (EIA/IS 632).
Subsystem	A system part that is derived from decomposing a system or another subsystem.
System	An integrated composite of people, products, and processes that provide a capability to satisfy a stated need or objective (EIA-IS-632).
System and software development	An inclusive term encompassing new development, modification, reuse, reengineering, maintenance, and other activities resulting in system/software products (Derived from MIL-STD-498).
System part	Any part of a decomposed system, including a subsystem, configuration item, component, or the system itself.
Tangible item	Developed items, such as hardware, that are physical entities. Items that are physical in nature, but contain embedded nontangible components. (An airplane or a car are tangible items even though they contain embedded software.)
Validate	Evaluate the results of the requirements analysis activities to ensure compliance with customer expectations and project and external constraints (Derived from IEEE P1220).
Verify	Evaluate the products of a given development phase to determine whether they satisfy the requirements specified at the start of the phase.

REFERENCES

Department of Commerce
1993

Draft Federal Information Processing Standards Publication 183: Announcing the Standard for INTEGRATION DEFINITION FOR FUNCTION MODELING (IDEF0). Washington, D. C.: Department of Commerce, National Institute of Standards and Technology, Computer Systems Laboratory.

Department of Defense
1994

Military Standard: Software Development and Documentation, MIL-STD-498. Washington, D. C.: Department of Defense.

EIA
1994

EIA Interim Standard—Systems Engineering. EIA/IS-632. Washington, D. C.: Electronic Industries Association.

IEEE
1994

IEEE P1220: Trial-Use Standard for Application and Management of the Systems Engineering Process (Final Draft). New York, New York: IEEE, Inc.

NCOSE
1995

Systems Engineering in the Global Marketplace: Proceedings of the Fifth Annual International Symposium of the National Council on Systems Engineering. St. Louis, Missouri.

ISO/IEC
1995

International Standard: Information Technology—Software Life Cycle Process. Geneva, Switzerland: ISO/IEC.

Office of Naval Research
1994

First Annual Workshop on Engineering of Systems in 21th Century: Facing the Challenge Proceedings. Dahlgren, Virginia: Office of Naval Research, Naval Surface Warfare Center.

Paulk, M.C., B. Curtis,
M.B. Chrissis, and C.V. Weber
1993

Capability Maturity Model for Software, version 1.1, CMU/SEI-93-TR24. Pittsburgh, Pennsylvania: Software Engineering Institute, Carnegie Mellon University.

Popkin Software and Systems, Incorporated
1991–1995

System Architect's Tool for Business Process Reengineering (SA/BPR). New York, New York: Popkin Software and Systems, Incorporated.

Software Engineering Institute
1994

SE-CMM Model Description Systems Engineering Improvement. SECMM-94-04, Release 2.03, November 8. Pittsburgh, Pennsylvania: Software Engineering Institute.

Software Productivity Consortium 1993a — *Reuse-Driven Software Processes Guidebook*, SPC-92019-CMC, version 02.00.03. Herndon, Virginia: Software Productivity Consortium.

1993b — *System Engineering Workshop Summary*, SPC-93128-MC, version 01.00.05. Herndon, Virginia: Software Productivity Consortium.

1994a — *ARPA/SISTO Software Strategy Workshop, August 18, 1994*, SPC-94092-CMC, version 01.00.01. Herndon, Virginia: Software Productivity Consortium.

1994b — *Process Engineering With the Evolutionary Spiral Process Model*, SPC-93098-CMC, version 01.00.06. Herndon, Virginia: Software Productivity Consortium.

1995a — *Minutes of ESIS User Group Meeting, October 19, 1995*. Herndon, Virginia: Software Productivity Consortium (Distributed via the World Wide Web).

1995b — *A Tailorable Process for Systems Engineering* SPC-94095-CMC, version 01.00.05. Herndon, Virginia: Software Productivity Consortium.